UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Otimização com Muitos Objetivos por Múltiplos Enxames Aplicada ao Escalonamento Dinâmico de Projetos de Software

Dissertação de Mestrado

Rodrigo Octávio Melo do Amaral

São Cristóvão – Sergipe

2018

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Rodrigo Octávio Melo do Amaral

Otimização com Muitos Objetivos por Múltiplos Enxames Aplicada ao Escalonamento Dinâmico de Projetos de Software

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Sergipe como requisito parcial para a obtenção do título de mestre em Ciência da Computação.

Orientador(a): Prof. Dr. André Britto de Carvalho
Coorientador(a): Profª. Drª. Leila Maciel de Almeida e Silva

São Cristóvão – Sergipe

2018

Rodrigo Octávio Melo do Amaral

Otimização com Muitos Objetivos por Múltiplos Enxames Aplicada ao Escalonamento Dinâmico de Projetos de Software/ Rodrigo Octávio Melo do Amaral. – São Cristóvão – Sergipe, 2018-

79 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. André Britto de Carvalho

Dissertação de Mestrado – UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO, 2018.

1. escalonamento de projetos. 2. search-based software engineering. 3. otimização com muitos objetivos. 4. múltiplos enxames. I. André Britto de Carvalho. II. Universidade Federal de Sergipe. III. Programa de Pós-Graduação em Ciência da Computação.

CDU 02:141:005.7

UNIVERSIDADE FEDERAL DE SERGIPE
PRÓ-REITORIA DE PÓS-GRADUAÇÃO E PESQUISA
COORDENAÇÃO DE PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Ata da Sessão Solene de Defesa da Dissertação do Curso de Mestrado em Ciência da Computação-UFS.
Candidato: RODRIGO OCTÁVIO MELO DO AMARAL

Em 31 dias do mês de Agosto do ano de dois mil e dezoito, com início às 13h00min, realizou-se na Sala de Seminário do DCOMP da Universidade Federal de Sergipe, na Cidade Universitária Prof. José Aloísio de Campos, a Sessão Pública de Defesa de Dissertação de Mestrado do candidato **RODRIGO OCTÁVIO MELO DO AMARAL**, que desenvolveu o trabalho intitulado: **" Otimização com Muitos Objetivos por Múltiplos Enxames Aplicada ao Escalonamento Dinâmico de Projetos de Software"**, sob a orientação do Prof. Dr. **André Britto de Carvalho**. A Sessão foi presidida pelo Prof. Dr. **André Britto de Carvalho** (PROCC/UFS), que após a apresentação da dissertação passou a palavra aos outros membros da Banca Examinadora, Prof. Dr. **Hendrik Teixeira Macedo** (PROCC/UFS) e, em seguida, ao Prof. Dr. **Márcio de Oliveira Barros** (UNIRIO). Após as discussões, a Banca Examinadora reuniu-se e considerou o mestrando (a) APROVADO "*(aprovado/reprovado)*" SEM "*(com/sem)*" ressalvas. Atendidas as exigências da Instrução Normativa 01/2017/PROCC, do Regimento Interno do PROCC (Resolução 67/2014/CONEPE), e da Resolução nº 25/2014/CONEPE que regulamentam a Apresentação e Defesa de Dissertação, e nada mais havendo a tratar, a Banca Examinadora elaborou esta Ata que será assinada pelos seus membros e pelo mestrando.

Cidade Universitária "*Prof. José Aloísio de Campos*", 31 de Agosto de 2018.

Prof. Dr. André Britto de Carvalho
(PROCC/UFS)
Presidente

Prof. Dr. Hendrik Teixeira Macedo
(PROCC/UFS)
Examinador Interno

Prof. Dr. Márcio de Oliveira Barros
(UNIRIO)
Examinador Externo

Rodrigo Octávio Melo do Amaral
Candidato

Para Catarina e Dante,
com todo amor.

Agradecimentos

Este trabalho não teria sido possível sem a orientação, o apoio e o encorajamento de várias pessoas que me acompanham nesta jornada. *"Eu sou porque nós somos"*.

Primeiramente, gostaria de agradecer ao meu orientador, Prof. André, pela parceria, confiança, paciência e por todo o intenso aprendizado durante a condução da orientação deste mestrado. Da mesma forma também agradeço a oportunidade de ter como co-orientadora a Prof. Leila, exemplo de trabalho e dedicação em sua história no DCOMP-UFS, pelo valioso apoio metodológico.

A todos os professores e colaboradores do PROCC-UFS, pelos conhecimentos transmitidos e por manterem a pós-graduação em Ciência da Computação viva e bem sucedida, apesar das dificuldades.

Aos colegas do Grupo de Pesquisa em Teoria e Otimização, em especial ao José Joaquim, por ter ajudado a desbravar o problema de pesquisa tema deste trabalho.

Aos pesquisadores Xiaoning Shen, Leandro Minku, Rami Bahsoon e Xin Yao pelo belo trabalho de pesquisa que inspirou esta dissertação e pelo desprendimento em disponibilizar os conjuntos de dados e insumos para os experimentos.

Aos colegas de trabalho do TRT da 20ª Região, por todo o incentivo a cursar este mestrado e todo o apoio para que essa meta fosse possível.

A meus pais, expressão divina de amor e serenidade, pela minha formação, por todo o zelo e por serem meu maior exemplo de vida.

A meus irmãos pelo companheirismo de todas as horas.

Por fim, gratidão sem medida à minha esposa Catarina e meu filho Dante, por toda a amorosa compreensão em suportar minhas constantes "ausências"durante todo esse período. Vocês dois fazem *tudo* valer a pena, sempre!

Viva como se fosse morrer amanhã.
Aprenda como se fosse viver para sempre.
(Mahatma Gandhi)

Resumo

Os processos da Engenharia de Software geralmente envolvem problemas com requisitos e restrições conflitantes. Para a solução desses problemas, recentemente surgiu o conceito de *Search Based Software Engineering* (SBSE). A SBSE consiste basicamente no uso de algoritmos de busca e otimização para encontrar soluções de forma automática e equilibrar o compromisso entre os objetivos dos problemas comuns a suas várias ramificações. Uma das áreas da SBSE ocupa-se do gerenciamento de projetos de software, sendo que um de seus principais desafios é o problema de escalonamento de projetos de software (SPSP, do inglês *Software Project Scheduling Problem*). Soluções para este problema buscam montar um cronograma de projeto de modo que a alocação dos empregados às tarefas disponíveis minimize tanto a duração total do projeto quanto seu custo final, caracterizando o SPSP como um problema de otimização multiobjetivo. No entanto, o ambiente de projetos de software está sujeito a muitas incertezas, o que faz com que o espaço de soluções possíveis se transforme ao longo do tempo. Essa natureza dinâmica traz a necessidade de que os cronogramas sejam também estáveis e robustos frente a mudanças, introduzindo novos objetivos a serem conciliados. Assim, nessa nova abordagem o SPSP é modelado como um problema de otimização dinâmica com muitos objetivos (quando há mais de três funções objetivo). Este trabalho tem como objetivo investigar a aplicação de algoritmos de otimização multiobjetivo ao problema de escalonamento dinâmico de projetos de software (DSPSP). Para isso, foi utilizada a meta-heurística de otimização por múltiplos enxames de partículas, abordagem ainda pouco explorada para aplicação ao DSPSP. O algoritmo proposto explora características do problema, buscando conciliar tanto o aspecto de otimização dinâmica, por meio do uso de múltiplas populações, quanto o de ser um problema com muitos objetivos, por meio dos métodos de arquivamento. Além da utilização de múltiplos enxames, o trabalho explora também o uso de mais dois algoritmos multiobjetivo já aplicados a outros problemas na área de SBSE, porém ainda não aplicados ao problema de SPSP dinâmico. Em conjunto com a otimização por múltiplos enxames, são exploradas ainda algumas estratégias heurísticas dinâmicas para a inicialização das populações, de forma a aproveitar características das melhores soluções já encontradas. Essas estratégias também são aplicadas aos demais algoritmos avaliados, a fim de verificar como elas influenciam o desempenho. A validação do algoritmo é feita por meio de um conjunto de experimentos que comparam o algoritmo proposto com dois algoritmos consagrados da literatura (NSGA-II e SMPSO).

Palavras-chave: escalonamento de projetos, search-based software engineering, otimização com muitos objetivos, múltiplos enxames

Abstract

Software engineering processes often involve problems with mutually conflicting requirements and constraints. To address these issues, the concept of Search-Based Software Engineering (SBSE) has emerged. SBSE consists on using search and optimization algorithms to balance the compromise between the objectives of the problems common to its various domains. One of these sub-areas deals with software project management, one of its main challenges being the Software Project Scheduling Problem (SPSP). Solutions to this problem aim to assemble a project schedule so that employee allocation to available tasks minimizes both the total project duration and its final cost. However, a software project environment is subject to many uncertainties, which makes the solution landscape change over time. This dynamic nature demands schedules to be stable and robust to changes, introducing new objectives to be reconciled into the problem. Because this dynamic formulation of SPSP is still under-explored, it is believed that it is possible to explore new meta-heuristics, incorporate features of the problem, and use many-objective optimization techniques to solve it. The objective of this work is to investigate how multiobjective algorithms can be applied to solve the Dynamic Software Project Scheduling problem (DSPSP). This work uses the meta-heuristic of optimization by multiple swarms of particles, an approach not yet explored when applied to DSPSP. The proposed algorithm also explores the problem characteristics, seeking to reconcile both the dynamic optimization aspect, by using multiple populations, as the fact it is many-objective problem, by using archiving methods. In addition to using multiple swarns, this work also explores the use of two multi-objective algorithms already applied to other problems in SBSE area, but not yet applied to DSPSP. In addition to multi-swarm optimization, some dynamic heuristic strategies for population initialization are explored to take advantage of the features of the best solutions already found. These strategies are also applied to the other evaluated algorithms in order to verify how they influence performance. The algorithm validation is done through a set of experiments which compared the proposed algorithm with two popular algorithms in literature (NSGA-II and SMPSO).

Keywords: project scheduling, search-based software engineering, many objective optimization, multiple swarm.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

DSPSP	Dynamic Software Project Scheduling Problem
MODPSP	Multi-Objective Dynamic Project Scheduling Problem
MOP	Multiobjective Optimization Problem
MaOP	Many-Objective Optimization Problem
MS2MO	Multi-Swarm Multi-Strategy Algorithm for Many-Objective Optimization
NSGA-II	Non-dominated Sorting Genetic Algorithm II
PSO	Particle Swarm Optimization
SBSE	Search-Based Software Engineering
SBSPM	Search-Based Software Project Management
SMPSO	Speed-constrained Multi-objective Particle Swarm Optimization

Sumário

1 Introdução

A Engenharia de Software pode ser definida como um conjunto de métodos, ferramentas e procedimentos que possibilita o controle do processo de desenvolvimento de software, estabelecendo princípios sólidos para que se obtenha software eficiente, confiável e economicamente viável (PRESSMAN, 2005). No entanto, persiste uma controvérsia se é adequado usar o termo "engenharia"aplicado a esse contexto. De acordo com Shaw (1990), engenharia envolve "a aplicação disciplinada de conhecimento científico para resolver restrições e requisitos conflitantes para problemas de significância prática e imediata". Todavia, o que se observa é que falta embasamento científico à prática da Engenharia de Software, que usualmente se baseia em soluções *ad hoc* e no conhecimento relativo a problemas passados.

Como tentativa de contemplar esse tipo de questionamento, percebe-se um aumento de interesse no estudo de técnicas de Engenharia de Software Baseada em Busca (SBSE, do inglês *Search Based Software Engineering*). Nessa abordagem, os problemas usuais da Engenharia de Software são reformulados como problemas de busca e otimização. Dessa forma, uma das características comuns aos demais ramos da engenharia é contemplada: encontrar soluções próximas do ótimo para problemas com requisitos e restrições conflitantes (HARMAN; JONES, 2001).

Partindo desse princípio, podemos considerar que as tarefas de alocação de recursos, escalonamento das atividades e estimativas de esforço para um projeto de software também podem ser candidatas a otimização por meio das técnicas da SBSE. De fato, segundo Ferrucci, Harman e Sarro (2014), mesmo antes da introdução do termo SBSE por Harman e Jones (2001), o Gerenciamento de Projetos de Software Baseado em Busca (SBSPM, do inglês *Search-Based Software Project Management*) já era tema de atividades de pesquisa. Em (CHANG et al., 1994), por exemplo, os autores tratam de escalonamento e alocação de recursos a projetos usando abordagens baseadas em busca.

Nesse contexto, um dos principais focos de pesquisa em SBSPM reside em definir quais

recursos devem ser utilizados para executar cada tarefa do projeto, assim como determinar o momento no qual cada tarefa deve ser executada. A literatura sobre o assunto engloba as mais diversas formulações e técnicas para investigar o problema, cada uma com ênfase em um ou mais fatores que envolvem o planejamento de projetos de software. Entre os estudos mais relevantes está o de Alba e Chicano (2007), no qual é concebida uma formulação que visa obter um cronograma que minimize a duração total do projeto, respeitando um conjunto de restrições e minimizando também o custo total do projeto. Trata-se de uma formulação que envolve a otimização de duas funções objetivo e que ficou conhecida pelo nome de SPSP (do inglês *Software Project Scheduling Problem*). Essa formulação define o SPSP como um problema de otimização multiobjetivo, que consiste em minimizar simultaneamente duas ou mais funções objetivo conflitantes entre si. Para solucionar esse tipo de problema, geralmente são empregados algoritmos evolutivos.

Contudo, os trabalhos sobre o tema costumam considerar que o ambiente de projetos de software seria estático, o que difere significativamente da realidade, na qual incertezas e eventos imprevistos interferem no planejamento. Mais recentemente, portanto, trabalhos como os de Xiao et al. (2010) e Shen et al. (2016), passaram a abordar formulações dinâmicas. Tais abordagens tratam o escalonamento e atribuição de tarefas em ambientes de projeto incertos e mutáveis ao longo do tempo, usando para isso métodos de otimização dinâmica multiobjetivo. Como consequência, outra característica explorada por Shen et al. (2016) é considerar um maior número de funções objetivo (4). Problemas de otimização com mais de 3 funções objetivo são definidos numa nova classe, chamada de Otimização com Muitos Objetivos (ISHIBUCHI et al., 2011) (SCHÜTZE; LARA; COELLO, 2011). Algoritmos evolucionários multiobjetivo tradicionais enfrentam dificuldades em resolver problemas com muitos objetivos. Recentemente diversas técnicas têm sido propostas resolver esse tipo problema.

Não obstante os avanços observados nas pesquisas sobre o tema, ainda há alguns pontos que merecem ser explorados em maior detalhe. Uma limitação importante é que as características próprias da formulação dinâmica do SPSP ainda são pouco exploradas pelos estudos encontrados na literatura na elaboração de algoritmos dedicados ao problema. Nessa mesma linha, as pesquisas sobre o tema também carecem de um enfoque que busque explorar técnicas para resolver problemas da Otimização com Muitos Objetivos.

Em (SHEN et al., 2016), além da proposta de uma nova formulação para o problema, é proposto ainda um algoritmo evolutivo adaptado essa formulação, chamado *dε-MOEA*, que busca incorporar diferentes técnicas próprias da resolução de problemas dinâmicos (NGUYEN; YANG; BRANKE, 2012). Além disso, apesar de propor quatro funções objetivo na formulação do problema, a abordagem não explora técnicas da otimização com muitos objetivos. Deve-se ressaltar ainda que Shen et al. (2016) apenas realizam estudo comparativo entre diferentes variações do próprio dε-MOEA, não existindo portanto avaliação do seu desempenho frente a outras meta-heurísticas.

Uma das finalidades da formulação do SPSP proposta por Shen et al. (2016) é buscar uma maior aproximação com ambientes de projetos de software reais, porém a solução proposta apresenta algumas limitações. As diferentes meta-heurísticas que vêm sendo aplicadas no contexto da SBSE apontam algumas alternativas para solucionar as limitações do modelo. Para contorná-las é possível explorar novas meta-heurísticas, incorporar características do problema e usar técnicas de muitos objetivos. Entre as meta-heurísticas possíveis está o uso de algoritmos genéticos e de técnicas de otimização por enxame de partículas (PSO, do inglês *Particle Swarm Optimization*). A técnica de PSO, por sua vez, pode ser estendida empregando mais de um enxame de partículas na busca. De acordo com Nguyen, Yang e Branke (2012), o uso de múltiplas populações é uma das alternativas para a garantia de diversidade em um ambiente de otimização dinâmico. Além disso, essa técnica vem sendo aplicada com sucesso para a resolução de problemas com muitos objetivos (BRITTO; MOSTAGHIM; POZO, 2013).

1.1 Objetivos

O objetivo deste trabalho é investigar a aplicação de algoritmos de otimização multi-objetivo ao problema de escalonamento dinâmico de projetos de software. Este objetivo geral divide-se nos seguintes objetivos específicos:

a) Explorar o uso de meta-heurísticas multiobjetivo, tais como algoritmos genéticos e PSO

b) Implementar algoritmo de otimização por múltiplos enxames específico para a formulação dinâmica do SPSP;

c) Fazer a análise de desempenho do algoritmo proposto em diferentes instâncias do problema de escalonamento de projetos de software.

1.2 Metodologia

Inicialmente, foi realizado um mapeamento sistemático como forma de delinear o estado da arte sobre o problema de escalonamento de projetos de software. Dessa forma, foi possível identificar na literatura as lacunas nos modelos disponíveis e descobrir quais técnicas vêm sendo utilizadas para resolvê-lo. Em seguida, foi implementado um *framework* para experimentos em SPSP que inclui a implementação do *Multi-Objective Dynamic Project Scheduling Problem* (MODPSP), modelo de escalonamento dinâmico de projetos proposto no trabalho de Shen et al. (2016). O modelo busca incorporar características de projetos reais, levando em consideração fatores como a incerteza nas estimativas do esforço necessário para uma tarefa, sobrecargas de comunicação em equipes grandes trabalhando em uma tarefa, níveis de proficiência da equipe nas habilidades requeridas pelo projeto, entre outras.

O *framework* desenvolvido neste trabalho incorpora a biblioteca *jMetal*, com o intuito de facilitar a aplicação de implementações consolidadas dos principais algoritmos e meta-heurísticas de otimização multiobjetivo. Este trabalho avaliou dois algoritmos que ainda não haviam sido explorados nos estudos sobre variantes dinâmicas do SPSP: NSGA-II e SMPSO.

Na sequência, foi desenvolvido um algoritmo de otimização por múltiplos enxames que permite explorar as características dinâmicas do problema. A algoritmo foi baseado no trabalho de Matos e Britto (2017), podendo ser parametrizado para utilizar estratégias heurísticas para reinicialização das populações a cada mudança no espaço de objetivos.

Como forma de avaliar o desempenho dos algoritmos, o *framework* foi utilizado para conduzir um conjunto de experimentos no qual são realizadas múltiplas simulações de projetos, de acordo com instâncias utilizadas no trabalho de Shen et al. (2016). Realizadas as simulações, os resultados são medidos e comparados em termos da métrica do hipervolume médio das soluções obtidas. Na avaliação dos resultados, o desempenho do algoritmo proposto foi comparado com o NSGA-II e o SMPSO. A comparação de desempenho do NSGA-II e SMPSO inclui ainda a avaliação do impacto da incorporação das estratégias heurísticas. As questões de pesquisa que guiaram os experimentos realizados neste trabalho estão definidas na subseção 1.2.1 a seguir.

1.2.1 Questões de Pesquisa

As questões de pesquisa (QP) que este estudo pretende investigar são as seguintes:

QP1. A quantidade de enxames influencia significativamente o desempenho do algoritmo proposto, de acordo com as métricas adotadas?

QP2. Como o desempenho do algoritmo proposto se compara ao de algoritmos da literatura quando aplicados ao DSPSP?

QP3. Qual a influência de cada estratégia heurística dinâmica no desempenho do algoritmo proposto?

1.3 Contribuições deste trabalho

O presente trabalho pretende apresentar as seguintes contribuições para o estudo do problema de escalonamento dinâmico de projetos:

- Desenvolvimento de um *framework* para experimentos em SPSP baseado na biblioteca jMetal, facilitando a elaboração de trabalhos futuros sobre o tema;
- Desenvolvimento de um algoritmo baseado em múltiplos enxames de partículas aplicado ao problema, como forma de investigar as possibilidades dessa meta-heurística em problemas de otimização dinâmica;

- Aplicação de NSGA-II e SMPSO ao escalonamento dinâmico de projetos, uma vez que estes ainda não haviam sido aplicados ao problema proposto;

1.4 Estrutura do Documento

Para facilitar a navegação e melhor entendimento, este documento está organizado em capítulos. O Capítulo 2 traz a fundamentação teórica e os trabalhos relacionados aos conceitos envolvidos no presente estudo. Em seguida, o Capitulo 3 descreve o problema de escalonamento dinâmico de projetos de software e propõe um algoritmo de otimização multiobjetivo baseado em múltiplos enxames de partículas para resolvê-lo. No Capítulo 4 são descritos os experimentos para validação da hipótese e seus resultados são discutidos. Por último, o Capítulo 5 apresenta as considerações finais e desafios futuros deste trabalho.

2

Fundamentação Teórica

Neste capítulo são apresentados os principais conceitos necessários para a compreensão desta dissertação. Na seção 2.1, apresentamos uma visão geral sobre o Problema de Escalonamento de Projetos de Software, situando-o no contexto da área de Engenharia de Software Baseada em Busca e, mais especificamente, como uma das tarefas da Gestão de Projetos Baseada em Busca (subseção 2.1.1). A seção 2.2 trata dos principais conceitos e técnicas envolvidos na Otimização Multiobjetivo, incluindo os princípios centrais da otimização dinâmica (subseção 2.2.3), que é tema deste trabalho.

2.1 Problema de Escalonamento de Projetos de Software

O termo *Search Based Software Engineering (SBSE)* foi usado pela primeira vez no trabalho de Harman e Jones (2001), que afirma a consolidação de um novo ramo de pesquisa e prática da Engenharia de Software. Essa nova direção é dada pela possibilidade de redefinir os problemas clássicos da Engenharia de Software na forma de problemas de busca em espaço de estados (HARMAN; JONES, 2001).

Uma característica comum aos diversos ramos da engenharia é a necessidade de otimizar o equilíbrio entre o uso de recursos e o desempenho da solução desejada. Da mesma forma, uma das preocupações da Engenharia de Software é tratar problemas com essas mesmas características: busca-se construir sistemas que sejam mais rápidos, baratos, confiáveis, escaláveis e adaptáveis, entre outros vários objetivos possíveis (FERRUCCI; HARMAN; SARRO, 2014). Em geral, tais objetivos precisam conciliar restrições conflitantes entre si, de maneira que soluções ótimas mostram-se praticamente impossíveis de encontrar (HARMAN; JONES, 2001). Portanto, técnicas baseadas em busca são candidatas naturais para solucionar tal categoria de problemas. Essas técnicas são muitas vezes derivadas da área de pesquisa operacional, tal como a programação linear, ou de meta-heurísticas como Algoritmos Genéticos (COLANZI et al.,

2013).

Técnicas baseadas em busca podem ser aplicadas nas mais diversas áreas da Engenharia de Software nas quais o conjunto de possibilidades de solução seja extremamente grande. Podemos desejar, por exemplo, descobrir qual o menor conjunto de casos de teste que cobre todas as ramificações de um programa, ou ainda qual o conjunto de requisitos de um software que apresenta o melhor equilíbrio entre custo de desenvolvimento e satisfação do cliente (HARMAN; MANSOURI; ZHANG, 2012). Nessa linha, cada domínio da Engenharia de Software possui particularidades que se prestam à aplicação de técnicas de busca e otimização, conforme detalhado a seguir.

- **Requisitos e Especificação.** O processo de Engenharia de Software tem como parte fundamental a engenharia de requisitos, responsável entre outras tarefas por definir, estimar, medir e priorizar os requisitos do software a ser desenvolvido. Nesse sentido, a SBSE tem sido aplicada na priorização de requisitos de modo a otimizar o esforço e a geração de valor para as partes interessadas (*stakeholders*) a cada liberação de versão do software. Um dos problemas de maior destaque nessa área é o *Next Realease Problem* (NRP), que busca encontrar um subconjunto das partes interessadas cujos requisitos devem ser satisfeitos, equilibrando sua importância com as restrições de recursos de desenvolvimento (HARMAN; MANSOURI; ZHANG, 2012).

- **Ferramentas e Técnicas para Projeto de Software.** A pesquisa sobre técnicas baseadas em busca para o projeto de software (*software design*) aborda desde o projeto de alto nível da arquitetura até a refatoração de software, passando pela melhoria e previsão da qualidade de software. O projeto de arquiteturas orientadas a objeto preocupa-se com conceitos tais como a atribuição de responsabilidades a classes, o acoplamento e a coesão entre os componentes, a composição de hierarquias, entre outros. Para conseguir boas soluções de projeto em um problema de atribuição de responsabilidades, por exemplo, é desejável que a localização de métodos e atributos esteja distribuído de maneira ótima entre as classes de um sistema (RÄIHÄ, 2010).

- **Verificação de Software.** A verificação de software diz respeito à tarefa de assegurar que o software implementado está de acordo com os requisitos especificados. Para este fim, técnicas de busca podem ser empregadas para explorar o espaço de estados dos programas. Trabalhos nessa área incluem o uso de algoritmos de checagem de modelos baseados em *Genetic Programming (GP)* (KATZ; PELED, 2008), *Ant Colony Optimization (ACO)* (ALBA; CHICANO, 2008), *Particle Swarm Optimization (PSO)* (MAHANTI; BANERJEE, 2006), entre outros.

- **Testes e Depuração.** A qualidade de um software pode ser avaliada por meio de processos de testes, que medem critérios de quanto o software construído é correto, completo, seguro, confiável, eficiente, portável, entre outros (HARMAN; MANSOURI; ZHANG,

2009). Com o crescente interesse dos pesquisadores nesse domínio, a aplicação de SBSE relacionada a testes chegou a responder por mais da metade dos trabalhos disponíveis na literatura sobre o tema em 2010 (HARMAN; MANSOURI; ZHANG, 2012). Todas as abordagens para a geração de casos de teste baseados em busca levam em consideração que o conjunto de casos de teste (contendo as entradas possíveis para um dado programa) pode ser considerado como um espaço de busca. Dessa forma, o critério de adequação do teste pode ser codificado como uma função de aptidão (HARMAN; MANSOURI; ZHANG, 2009).

- **Distribuição, Manutenção e Melhoria.** O ciclo de vida de um software envolve atividades voltadas para sua melhoria, correção e incremento de funcionalidades. No contexto da aplicação de SBSE nesse domínio, dois ramos de pesquisa têm maior destaque: modularização e refatoração. Em linhas gerais, modularização diz respeito a como agrupar os módulos de um software de modo a maximizar a coesão e minimizar o acoplamento entre eles, de acordo com determinada granularidade. Refatoração é o processo de aplicar melhorias à estrutura interna de um programa sem alterar o seu comportamento e semântica (HARMAN; MANSOURI; ZHANG, 2012).

- **Gestão de Projetos de Software.** As tarefas de alocação de recursos, escalonamento das atividades e estimativas de esforço para um projeto de software também podem ser otimizadas por meio da aplicação das técnicas da SBSE. Os esforços de pesquisa em SBSE aplicada à gestão de projetos de software concentram-se principalmente em duas áreas: planejamento de projetos por meio da otimização de uso dos recursos e estimativa de custos através da criação de modelos preditivos (HARMAN; MANSOURI; ZHANG, 2012). Por ser o foco deste trabalho, mais detalhes sobre este domínio da SBSE são apresentados na seção 2.1.1.

2.1.1 Gestão de Projetos de Software Baseada em Busca

Segundo o PMI (2004), um projeto é *"um esforço temporário empreendido para criar um produto, serviço ou resultado exclusivo"*. A disciplina de gestão de projetos, por sua vez, é *"a aplicação de conhecimentos, habilidades e técnicas para a execução de projetos de forma efetiva e eficaz"*. Quando se trata de projetos de desenvolvimento ou manutenção de software, a gestão de projetos pode ser contextualizada como uma atividade de Engenharia de Software. A execução de um projeto envolve a integração de diferentes áreas de conhecimento que devem atender a critérios específicos. Áreas como a gestão de custos, gestão do tempo ou gestão da qualidade, entre outras, trabalham com decisões que envolvem a conciliação entre enormes quantidades de possibilidades conflitantes entre si. Portanto, como vimos no ínicio desta seção 2.1, a gestão de projetos de software é uma das áreas que se presta à aplicação de técnicas de busca e otimização para aprimorar seus resultados. Como exemplo de aplicação, podemos citar as atividades de estimativa de custos, planejamento do projeto e gerenciamento da qualidade,

que envolvem encontrar um ponto de equilíbrio entre objetivos concorrentes e potencialmente conflitantes (FERRUCCI; HARMAN; SARRO, 2014). Dessa forma, o grande número de dimensões e restrições de mercado tornam a gestão de projetos de software uma tarefa especialmente complexa, muitas vezes envolvendo o gerenciamento de grandes equipes um ambiente dinâmico e instável (PEIXOTO; MATEUS; RESENDE, 2014). Apesar da complexidade envolvida, a produção de software constitui um mercado cada vez mais competitivo, no qual o *time-to-market* é crucial. Portanto, é importante otimizar cada aspecto dos projetos, com atenção especial para a sua duração.

Em particular, há o problema de definir quais recursos devem ser utilizados para executar cada tarefa do projeto, assim como determinar o momento no qual cada tarefa deve ser executada. De fato, a primeira aplicação de SBSE à Gestão de Projetos de Software foi proposta para o escalonamento de projetos e alocação de recursos (FERRUCCI; HARMAN; SARRO, 2014). Alba e Chicano (2007) definiram uma das formulações mais famosas desse problema, batizada de Problema de Escalonamento de Projetos (*Project Scheduling Problem - PSP*). Por ser aplicada a projetos de software, essa formulação também é referenciada na literatura como *Software Project Scheduling Problem - SPSP*, nomenclatura utilizada no presente trabalho.

Pode-se considerar o SPSP como uma extensão do Problema de Escalonamento de Tarefas com Restrição de Recursos (*Resource-Constrained Project Scheduling Problem - RCPSP*), uma vez que ambos preocupam-se em obter um cronograma que minimize a duração total do projeto, respeitando um conjunto de restrições (MERKLE; MIDDENDORF; SCHMECK, 2002). No entanto, o SPSP considera ainda o objetivo adicional de minimizar também o custo total do projeto, calculado a partir do custo associado a cada um dos membros da equipe. Além disso, o SPSP trata apenas de um tipo de recurso, que são os membros da equipe, porém considerando que cada um deles conta com diferentes conjuntos de habilidades e pode estar dedicado parcialmente a mais de uma tarefa (ALBA; CHICANO, 2007).

No SPSP, portanto, consideramos que os recursos são pessoas com um conjunto de *habilidades* e que recebem um *salário* para desempenhar as *tarefas* que compõem um projeto de software. Nesta pesquisa, assim como no trabalho de Alba e Chicano (2007), as pessoas que fazem parte da equipe do projeto serão chamadas de *empregados*. Na maioria dos projetos as tarefas são interdependentes entre si, o que implica a existência de uma relação de predecessores e sucessores entre as tarefas. Em suma, algumas tarefas do projeto têm como condição para seu início que outras tarefas estejam concluídas. Os conjuntos de empregados e tarefas, cada um com seus respectivos atributos, constituem os dados de entrada do SPSP.

A equipe do projeto (figura 1) é formada pelos empregados que são alocados para exercer pelo menos uma tarefa. Os atributos de cada empregado são:

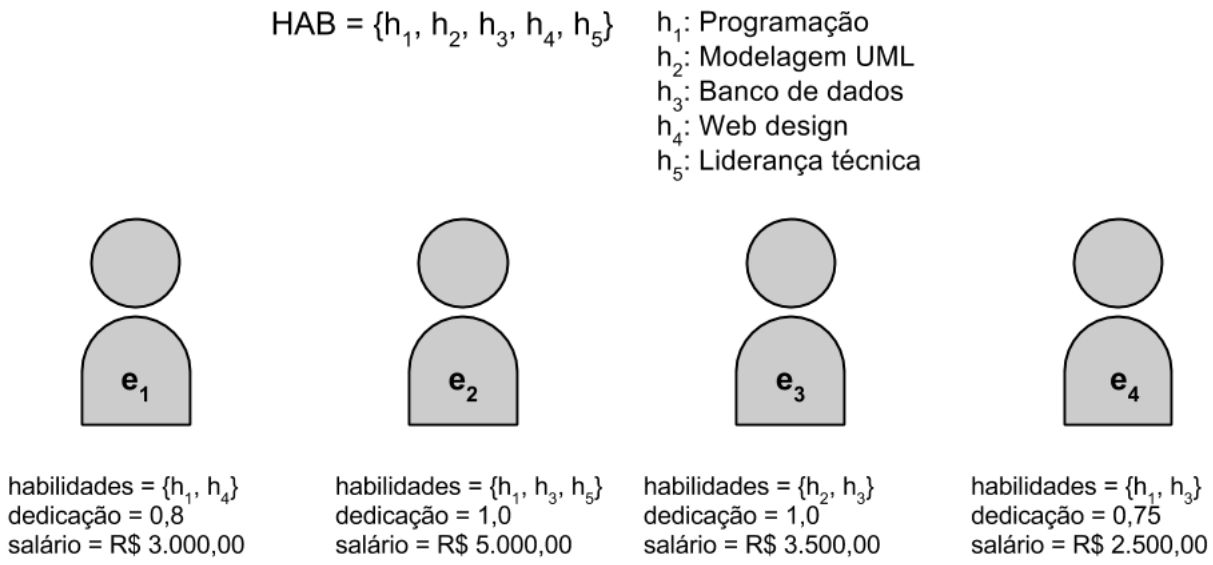
(1) o conjunto de habilidades que ele possui;

(2) a dedicação máxima ao projeto e

(3) o valor do seu salário mensal.

As habilidades de um empregado são as áreas de competência nas quais ele pode atuar. Por exemplo, um determinado empregado pode estar qualificado para atuar com modelagem UML e bancos de dados, enquanto outro pode ser especialista em programação e *web design*. A dedicação máxima de um empregado a um projeto é definida como a razão entre as horas do dia dedicadas ao projeto e a duração do expediente do empregado.

Figura 1 – Exemplo de equipe de empregados em um projeto



Fonte: Adaptado de Alba e Chicano (2007).

As tarefas do projeto (figura 2) são unidades ou pacotes de trabalho suficientemente bem delimitados para os quais se conhecem

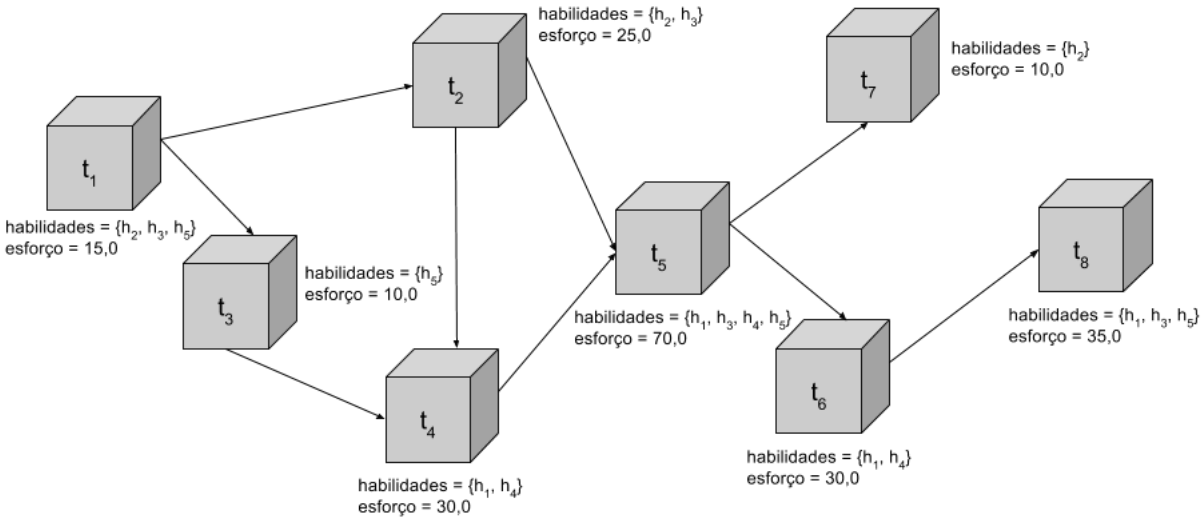
(1) o conjunto de habilidades necessárias para sua execução e

(2) o esforço estimado para completá-lo.

Na definição original de Alba e Chicano (2007), o esforço requerido para uma tarefa é medido em homem-mês (HM), que denota o tempo que um único empregado levaria para completar a tarefa caso trabalhasse nela sozinho.

A relação de dependência entre as tarefas é modelada por meio de uma estrutura conhecida como Grafo de Precedência de Tarefas (*Task Precedence Graph - TPG*). Trata-se de um grafo acíclico dirigido $G(V,A)$, no qual os vértices $V = \{t_1, t_2, ..., t_T\}$ representam as tarefas e as arestas $(t_i, t_j) \in A$ indicam que a tarefa t_i deve necessariamente ser concluída antes que a tarefa t_j seja iniciada.

Figura 2 – Precedência de tarefas em um projeto



Fonte: Adaptado de Alba e Chicano (2007).

Os objetivos do SPSP são minimizar o custo e a duração do projeto. Podemos perceber que, a princípio, estamos tratando de dois objetivos conflitantes entre si. Em geral, quando se deseja diminuir os custos de um projeto, aloca-se menos recursos, o que via de regra acarreta um tempo maior para a conclusão das tarefas. O contrário também é verdadeiro: quando se quer reduzir o prazo estimado de encerramento do projeto, tenta-se alocar mais recursos para ele, aumentando os custos. Tais características, portanto, classificam o SPSP como um problema de otimização multiobjetivo, para o qual podem ser aplicadas as técnicas de solução disponíveis na literatura.

Além dos objetivos concorrentes, o SPSP estabelece ainda algumas restrições que devem ser observadas, sob pena de tornar a solução inválida (não-factível):

1. Cada tarefa deve ser executada por pelo menos um empregado;
2. O conjunto de habilidades requeridas por uma tarefa deve pertencer ao conjunto formado pela união das habilidades dos empregados alocados para a tarefa;
3. Nenhum empregado deve exceder a sua dedicação máxima ao projeto, ou seja, não deve haver horas extras.

Sabendo quais são os componentes de uma instância do SPSP, podemos representar uma solução por meio de uma matriz $X = (x_{ij})$ de tamanho $E \times T$ onde $x_{ij} \geq 0$. Cada elemento x_{ij} indica o grau de dedicação do empregado a uma tarefa t_j. Dessa forma, por exemplo, se um empregado e_i tem dedicação 1 a uma tarefa t_j, significa que seu expediente é totalmente dedicado

a ela. Caso o valor do grau de dedicação seja 0, isto indica que o empregado não está alocado à tarefa em questão. A tabela 1 ilustra um exemplo de solução candidata do SPSP na forma de matriz de dedicação.

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
e_1	0,00	0,00	0,00	0,80	0,40	0,20	0,00	0,50
e_2	1,00	0,80	0,75	0,25	0,80	0,20	0,00	0,35
e_3	0,50	0,60	0,00	0,00	1,00	0,00	1,00	0,40
e_4	0,75	0,45	0,00	0,60	0,30	0,75	0,00	0,50

Tabela 1 – Exemplo de solução do SPSP codificada como uma matriz de dedicação

Uma vez conhecida a matriz de dedicação X, é possível calcular a duração de cada tarefa, bem como seus tempos de início e fim. Para isso, tomamos o esforço estimado para a tarefa t_j e dividimos pelo somatório do grau de dedicação de cada empregado à tarefa (equação 2.1). A partir daí, usando as durações individuais das tarefas e o TPG podemos obter a duração total do projeto.

$$t_j^{duracao} = \frac{t_j^{esforco}}{\sum_{i=1}^{E} x_{ij}} \tag{2.1}$$

Para obter o custo total do projeto, basta então somar os pagamentos de salários ao empregados de acordo com a dedicação de cada um ao projeto. Isso é feito multiplicando o salário mensal do empregado pelo tempo gasto por ele no projeto, que por sua vez é obtido somando as dedicações e multiplicando pela duração de cada tarefa (equação 2.2).

$$p_{custo} = \sum_{i=1}^{E} \sum_{i=1}^{T} e_i^{salario} \cdot x_{ij} \cdot t_j^{duracao} \tag{2.2}$$

2.1.2 Trabalhos Relacionados

O uso de Algoritmos Genéticos aplicado a problemas da área de gerenciamento de projetos havia sido explorado anteriormente por Chao (1995) e Chang, Christensen e Zhang (2001), apresentado a abordagem *Software Project Management Net (SPM-Net)* e lançando as bases para a exploração do uso da técnica de AG para alocação e escalonamento de recursos. Técnicas de *Hill Climbing*, Algoritmos Genéticos e *Simulated Annealing* (SA) foram utilizadas nos trabalhos de Antoniol, Di Penta e Harman (2004) e Antoniol, Penta e Harman (2005), nos quais investigou-se o problema da alocação de pacotes de trabalho com o objetivo específico de reduzir a duração do projeto.

Em seguida, conforme mencionado na subseção 2.1.1, Alba e Chicano (2007) definiram uma das formulações mais populares para o problema de escalonamento de projetos de software, o SPSP. O trabalho apresenta uma abordagem que utiliza Algoritmos Genéticos para aproximar

a solução do problema. O algoritmo utilizado procura minimizar os objetivos de duração e custo do projeto, tendo como entrada informações sobre os membros da equipe, as tarefas do projeto e a relação de dependência entre elas. A função de aptidão proposta por Alba e Chicano (2007) é constituída por uma soma ponderada simples para calcular a qualidade de uma solução levando em conta a duração e custo do projeto. Nela, os pesos têm a finalidade de permitir que os gerentes de projeto moldem a aptidão de acordo com suas necessidades.

Posteriormente, Chicano et al. (2011) estenderam o trabalho para lidar com múltiplos objetivos por meio de abordagem baseada em otimização de Pareto, utilizando algoritmos clássicos de otimização multiobjetivo como NSGA-II, SPEA2, PAES, MOCell e GDE3.

Alguns trabalhos dedicam-se a estudar as limitações da formulação original do SPSP, em especial a questão do quanto as soluções são robustas frente a mudanças nos parâmetros do problema. Antoniol, Di Penta e Harman (2004) observam que os parâmetros são baseados em estimativas que podem estar erradas ou com um elevado grau de incerteza, devido a fatores como o retrabalho (comum em desenvolvimento de software) ou até mesmo o abandono de pacotes de trabalho pelos mais diversos motivos. Usando dados de um projeto real, Antoniol, Penta e Harman (2005) estudam as técnicas baseadas em busca aplicadas ao planejamento de projetos de manutenção de grande porte, que têm como características a rigidez nos prazos e o alto risco de afetar sistemas em produção na organização. Luna et al. (2014) abordam a escalabilidade das técnicas de otimização multiobjetivo face ao crescimento no tamanho dos projetos de software, que implica em espaços de busca cada vez maiores.

Ainda dentro do campo gerenciamento de projetos baseado em busca, há a aplicação de técnicas de otimização para a modelagem preditiva de projetos, mais notadamente na tarefa de estimativa de esforço e custo. Harman, Mansouri e Zhang (2012) estudam detalhes do problema de estimativa de custo de software, definindo representação, função de aptidão e algoritmo. Ferrucci et al. (2010) apresentam um apanhado dos principais estudos sobre estimativas de esforço de desenvolvimento de software baseadas em busca, elencando os trabalhos mais significativos e apontando direções de pesquisa futura. Sarro (2011) explora como o uso de abordagens multiobjetivo pode aperfeiçoar a efetividade dos métodos de estimativa de esforço baseados em busca.

Além da meta-heurística de Algoritmos Genéticos tradicional, há trabalhos que abordam o problema de escalonamento de projetos usando diversas outras técnicas. Alvarez-Valdes et al. (2006) desenvolvem um algoritmo baseado em *Scatter Search* para problemas de escalonamento de projetos baseados em recursos parcialmente renováveis. Esse tipo de recurso não é consumido ao longo do projeto, mas só se encontra disponível durante um período limitado de tempo. Heričko, Živkovič e Rozman (2008) buscaram descobrir como otimizar o tamanho de equipe de projeto em função do tamanho do software e do esforço de desenvolvimento, usando para isso um algoritmo de otimização baseado em gradiente. Kang, Jung e Bae (2011) estudaram a alocação de recursos humanos por meio do uso do algoritmo *Accelerated Simulated Annealing*

(ASA), uma variação do SA, considerando uma série de restrições que, quando violadas, impõem penalidades às soluções. Chen e Zhang (2013) utilizaram a técnica de Otimização por Colônia de Formigas (ACO) para em conjunto com um escalonador baseado em eventos para ajustar a alocação de empregados na ocorrência de eventos de adição ou remoção de membros da equipe. Wu et al. (2016) propõem a adoção de uma hiper-heurística evolutiva para resolver o SPSP. Essa abordagem permite buscar pela melhor heurística ao invés de buscar a melhor solução. Para isso, o método permite escolher tanto o operador de mutação quanto o de cruzamento.

Um desafio que se impõe aos pesquisadores da área é a falta de dados de projetos do mundo real. Essa limitação perdura até os dias atuais, o que faz com que a maioria dos estudos recorra ao uso de dados sintéticos para realizar seus experimentos (FERRUCCI; HARMAN; SARRO, 2014). Como uma das exceções, pode-se mencionar o estudo empírico de Di Penta, Harman e Antoniol (2011), que aborda o uso de técnicas baseadas em busca combinadas a um modelo de simulação de filas aplicados a dados obtidos de dois projetos reais de manutenção de software comercial de larga escala.

Outra característica comum a muitos trabalhos de pesquisa é o uso de algoritmos de propósito geral aplicados ao SPSP. São poucos os estudos que buscam explorar as particularidades específicas do problema para adaptar novas formulações de algoritmos dedicados, a fim de obter algum incremento em desempenho.

Por fim, ao contrário do que grande parte dos estudos supõe, a gestão de cronogramas em projetos de software reais geralmente ocorre em ambientes dinâmicos e incertos. Portanto, é necessário que as abordagens para o problema de escalonamento de tarefas em projetos de software levem em consideração essas características dinâmicas. O escalonamento dinâmico de projetos de software é descrito mais detalhadamente na seção 2.1.3.

2.1.3 Escalonamento dinâmico de projetos de software

A formulação do SPSP baseia-se no fato de que sua solução depende de conhecimento previamente obtido em projetos anteriores. A precisão de informações tais como tarefas identificadas, dependências entre tarefas e esforço estimado geralmente dependem da experiência do gerente de projeto. Portanto, supõe-se que essas informações são sempre conhecidas de antemão e permanecem as mesmas durante o curso do projeto.

Todavia, os projetos do mundo real estão sujeitos a toda sorte de ocorrências difíceis de prever. Devido à natureza flexível do software, projetos de desenvolvimento e manutenção são especialmente sensíveis a mudanças, uma vez que a reformulação dos requisitos por parte dos clientes é algo corriqueiro e normal.

Considerando essa realidade, o trabalho de Xiao et al. (2010) leva em conta situações disruptivas que podem levar ao replanejamento de cronogramas de projetos, propondo um método multiobjetivo para calcular as reprogramações por meio de algoritmos genéticos.

Posteriormente, Shen et al. (2016) estendem a formulação do SPSP tradicional considerando as incertezas e os eventos dinâmicos que ocorrem com frequência durante o desenvolvimento de projetos de software, de modo que seja necessário reprogramar o cronograma do projeto. Para contemplar essa formulação, foi elaborado um modelo matemático para o problema multiobjetivo de escalonamento dinâmico de projetos, batizado de *MultiObjective Dynamic Project Scheduling Problem (MODPSP)* (SHEN et al., 2016). Nesse modelo, são considerados quatro objetivos na solução do problema:

1. custo total do projeto;
2. duração do projeto;
3. robustez do cronograma;
4. estabilidade do cronograma.

Dessa forma, temos mais dois objetivos em relação à formulação do SPSP descrita por Alba e Chicano (2007). *Robustez* é a medida do quanto um cronograma é sensível em relação às incertezas decorrentes da estimativa de esforço das tarefas. *Estabilidade* mede o quanto um novo cronograma difere em relação ao cronograma original. Esse modelo é tema de estudo nesta dissertação e será discutido com mais detalhes no Capítulo 3.

2.2 Otimização Multiobjetivo

Nesta seção são apresentados conceitos relativos a problemas de otimização, destacando os fundamentos da otimização multiobjetivo (subseção 2.2.2) e das estratégias para resolução problemas de otimização dinâmica (subseção 2.2.3).

2.2.1 Otimização

Em termos gerais, otimizar significa encontrar um conjunto de soluções que representam os melhores valores possíveis para contemplar um determinado problema. A medida dessa qualidade é dada por uma ou mais funções objetivo, cujos valores de retorno são avaliados por um tomador de decisão quanto a sua adequação ao problema.

Uma definição formal genérica para um problema de otimização consiste em minimizar (ou maximizar) $f(\vec{x})$ sujeita a $g_i(\vec{x}) \leq 0$, $i = \{1, \cdots, k\}$ e $h_j(\vec{x}) = 0$, $j = \{1, \cdots, p\}$ $\vec{x} \in \Omega$. Uma solução minimiza (ou maximiza) um escalar $f(\vec{x})$ onde $\vec{x}$ é um vetor de decisão variável n-dimensional $\vec{x} = (x_1, \cdots, x_n)$ em um universo Ω (COELLO et al., 2007).

Na definição acima, $g_i(\vec{x}) \leq 0$ e $h_j(\vec{x}) = 0$ representam restrições que precisam ser satisfeitas ao otimizar $f(\vec{x})$. O conjunto Ω contém todos os valores possíveis de $\vec{x}$ que podem ser usados para satisfazer $f(\vec{x})$ e suas restrições.

2.2.2 Otimização Multiobjetivo

Um Problema de Otimização Multi-Objetivo (MOP, do inglês *Multi-Objective Optimization Problem*) tem por finalidade satisfazer duas ou mais funções objetivo simultaneamente. Essas funções mapeiam objetivos que são conflitantes entre si, ou seja, quando o valor de uma delas melhora, o valor de alguma outra piora. Tal característica significa que não há apenas uma única melhor solução, mas sim um conjunto delas. O conjunto das melhores soluções de um problema de otimização multiobjetivo é obtido por meio da Teoria de Otimalidade de Pareto (CARVALHO, 2013).

Uma solução de um problema de otimização é representada pelo vetor $\vec{x} = (x_1, x_2, \cdots, x_n)$ pertencente a Ω, que é o conjunto de soluções possíveis para o problema. A qualidade da solução de um problema de otimização é definida por critérios expressos por funções computáveis das variáveis de decisão, representadas por $f_j(\vec{x})$.

No caso de problemas multiobjetivo, existem várias funções objetivo. Sendo assim, usa-se um vetor de funções objetivo $\overrightarrow{f(\vec{x})} \in \Lambda$, onde Λ é o espaço do vetor das funções objetivo. Um problema multiobjetivo pode ser definido por:

$$Minimize \overrightarrow{f(\vec{x})} = (f_1(\vec{x}), f_2(\vec{x}), \cdots, f_m(\vec{x}))$$

A finalidade da otimização multiobjetivo é otimizar as m funções objetivo de maneira simultânea, de modo a encontrar um conjunto de soluções que representam o melhor compromisso entre os objetivos. Para tanto, existem alguns conceitos fundamentais a várias técnicas de otimização multiobjetivo, conforme segue:

Dominância de Pareto. O conceito de dominância de uma solução por outra é essencial na composição dos métodos de otimização multiobjetivo. Quando se deseja resolver um problema de minimização, diz-se que um vetor de objetivos $\overrightarrow{f(\vec{x})} = (f_1(\vec{x}), \cdots, f_m(\vec{x}))$ domina outro vetor $\overrightarrow{f(\vec{y})} = (f_1(\vec{y}), \cdots, f_m(\vec{y}))$ se e somente se $\overrightarrow{f(\vec{x})}$ é menor ou igual a $\overrightarrow{f(\vec{y})}$ em todas as dimensões e é menor em pelo menos uma, ou seja, $\forall_i \in \{1, \cdots, m\}$, $f_i(\vec{x}) \leq f_i(\vec{y})$ e $\exists_i \in \{1, \cdots, m\} : f_i(\vec{x}) < f_i(\vec{y})$. O símbolo $\prec$ denota relação de dominância do predecessor sobre o sucessor. Quando se escreve $\overrightarrow{f(\vec{x})} \prec \overrightarrow{f(\vec{y})}$ isso quer dizer que $\overrightarrow{f(\vec{x})}$ *domina* $\overrightarrow{f(\vec{y})}$. A figura 3 ilustra um exemplo para o caso de minimização de dois objetivos.

Otimalidade de Pareto. Uma solução $\vec{x} \in \Omega$, onde $\vec{u} = F(\vec{x}) = f_1(\vec{x}), \cdots, f_k(\vec{x})$ é chamada de *ótimo de Pareto* quando não existe solução $\vec{x'} \in \Omega$, onde $\vec{v} = F(\vec{x'}) = f_1(\vec{x'}), \cdots, f_k(\vec{x'})$ tal que $\vec{u} \prec \vec{v}$.

Conjunto Ótimo de Pareto. Quando se trata de um problema multiobjetivo $F(x)$, há mais de uma solução ótima de Pareto. Portanto, um *conjunto ótimo de Pareto* P^* é assim definido: $P^* = \{\vec{x} \in \Omega | \neg \exists \vec{x'} \in \Omega\}, F(\vec{x'}) \prec F(\vec{x})$

Fronteira de Pareto. Dado um conjunto ótimo de Pareto P^* em um problema multiob-

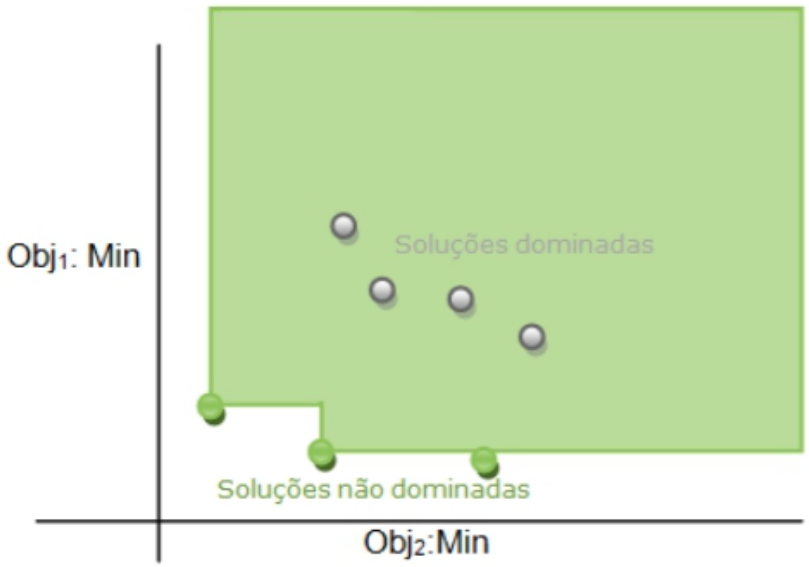


Figura 3 – Relação de dominância de Pareto em para um espaço bi-objetivo (extraído de Carvalho (2013))

jetivo, a *fronteira de Pareto* é definida como $PF^* = \{\vec{u} = F(\vec{x}) | \vec{x} \in P^*\}$. Dessa forma, uma vez conhecida a Fronteira de Pareto, o tomador de decisão pode escolher as soluções desejadas de acordo com o desempenho obtido para cada objetivo.

Considerando que o objetivo de um algoritmo de otimização baseado em meta-heurísticas é encontrar um conjunto de soluções que se aproxima do conjunto de Pareto ótimo, então a esse conjunto dá-se o nome de *conjunto aproximado*. A imagem do conjunto aproximado no espaço de objetivos é denominada *fronteira de Pareto aproximada*.

Algoritmos populacionais são usados para resolver problemas de otimização multiobjetivo atuando de duas maneiras: a primeira consiste na incorporação um mecanismo de seleção que, via de regra, baseia-se nos conceitos da Otimalidade de Pareto e segunda é a adoção de mecanismos que buscam preservar a diversidade, evitando a convergência para uma só solução. Devido à natureza complexa dos MOPs, os algoritmos populacionais têm como foco determinar uma fronteira de Pareto aproximada no espaço de objetivos (CARVALHO, 2013).

A *convergência* da fronteira de Pareto aproximada em direção da fronteira de Pareto real é a aproximação do conjunto de soluções obtido até iteração corrente do algoritmo até a fronteira de Pareto real. Quando menor a distância entre essas fronteiras, maior é a convergência do conjunto de soluções. O conceito de *diversidade*, por sua vez, diz respeito ao quanto as soluções obtidas estão espalhadas pelo espaço de busca, especialmente em relação à fronteira de Pareto. Uma boa diversidade significa que as soluções estão dispersas por toda a região da fronteira de Pareto, ao invés de concentradas ao redor de uma área restrita.

2.2.3 Otimização Dinâmica

Grande parte dos problemas do mundo real não apenas são compostos por múltiplos objetivos concorrentes a serem satisfeitos, como também possuem características que mudam ao longo do tempo. Além disso, existe a incerteza inerente a variáveis, restrições, coeficientes e até mesmo aos próprios objetivos. Dessa forma, alternativas de solução que eram inadequadas no passado podem ter se tornado interessantes no presente, e vice-versa. Problemas com essas características são conhecidos como Problemas de Otimização Dinâmica (*Dynamic Optimization Problems - DOPs*) (CRUZ; GONZÁLEZ; PELTA, 2011).

O caráter dinâmico dessa classe de problemas é dado pela ocorrência de *eventos* que modificam suas características. Exemplos de eventos do mundo real que podem afetar as instâncias de um problema são a chegada de novas tarefas em um projeto, demissão/admissão de funcionários, mudanças no cenário econômico-financeiro, defeitos em equipamentos, variação na disponibilidade de recursos, entre outros (NGUYEN; YANG; BRANKE, 2012).

Na definição de Cruz, González e Pelta (2011), um problema de otimização dinâmica é aquele que busca otimizar uma função $f(x,t)$ sujeita à restrição $x \in F(t) \subseteq S, t \in T$, onde:

- $S \in R^n$, sendo S o espaço de busca.
- t é o tempo.
- $f : S \times T \rightarrow R$, é a função objetivo que atribui um valor numérico ($f(x,t) \in R$) a cada possível solução ($x \in S$) no tempo t.
- $F(t)$ é o conjunto de soluções factíveis $x \in F(t) \subseteq S$ no tempo t.

Dessa forma, o que $F(t)$ indica é que um problema de otimização dinâmica é um problema no qual as restrições variam com o tempo. Sendo assim, o foco desse tipo de problema muda: não é suficiente apenas localizar uma solução ótima estacionária, mas deve-se também rastrear seu movimento no espaço de estados e no tempo.

Para abordar os DOPs, ferramentas da computação evolutiva e da inteligência de enxames têm sido usadas com resultados promissores. A grande vantagem dessas técnicas é sua inspiração nos processos da natureza, em seus sistemas auto-organizáveis e na evolução biológica, intrinsecamente sujeitos a ambientes incertos e instáveis (NGUYEN; YANG; BRANKE, 2012).

Em otimização dinâmica, assume-se que o estado do problema após a ocorrência de uma mudança tem relação estreita com seu estado antes da mudança. Dessa forma, um algoritmo de otimização deveria aprender com as buscas passadas com a finalidade de avançar de maneira mais efetiva. Nguyen, Yang e Branke (2012) sumarizam algumas alternativas para efetuar esse aprendizado que são comuns a vários estudos encontrados na literatura:

- **Detecção de mudanças.** As abordagens de otimização dinâmica muitas vezes tomam ações explícitas para responder a mudanças no ambiente. Essa mudanças são detectadas tanto pela reavaliação de soluções obtidas, quanto pelo comportamento do algoritmo. Na primeira alternativa, algumas soluções específicas chamadas de *detectores* são reavaliadas em busca de mudanças nos valores de suas funções objetivo ou em sua viabilidade. Os detectores podem ser uma parte da população, ou mantidos separadamente como uma subpopulação. O uso de detectores tem a vantagem de permitir uma detecção mais robusta, porém ao custo de ter que reavaliá-los a cada geração. No caso da detecção por comportamento do algoritmo, as mudanças são percebidas por meio do monitoramento da queda do valor médio das melhores soluções ao longo de certo número de gerações. A vantagem dessa alternativa é não requerer avaliações adicionais das soluções, mas por não haver detectores dedicados, não se pode garantir que as mudanças sejam detectadas.

- **Introdução de diversidade na ocorrência de mudanças.** Ao contrário do que ocorre em problemas estáticos, na otimização dinâmica a convergência do algoritmo não é necessariamente algo desejável. Isso se deve à possibilidade de que mudanças no espaço de soluções possam ocorrer em uma área em que não existam membros do algoritmo, falhando em detectar o ótimo global móvel. Portanto, uma possível solução para isso é incrementar a diversidade após a detecção de cada mudança. Por não precisar desperdiçar recursos na manutenção da diversidade o tempo inteiro, mas apenas na ocorrência de mudanças, essa abordagem pode concentrar-se inteiramente na busca. No entanto, existem as desvantagens de precisar saber da ocorrência de mudanças, sejam elas fáceis de detectar ou não, além da dificuldade de identificar o tamanho correto da mutação necessária.

- **Manutenção da diversidade durante a busca.** Também relacionada à diversidade das soluções, uma outra abordagem é manter a diversidade da população ao longo toda a busca, evitando que toda ela convirja para um único lugar. Esse tipo de método não precisa detectar as mudanças explicitamente, pois baseia-se na diversidade para adaptar-se a elas. Tal comportamento pode ser obtido através de várias abordagens, seja por meio da adição de indivíduos aleatórios à população, da colocação de indivíduos em locais específicos, de aprendizagem incremental, entre outras. Este método apresenta bom desempenho em problemas nos quais ocorrem grandes mudanças no espaço de busca e nos quais elas ocorrem raramente. Todavia, um efeito colateral do método é a sua lentidão, além de não ser efetivo quando as mudanças são pequenas.

- **Adição de componentes de memória aos algoritmos.** Nos casos em que as mudanças no espaço de busca são periódicas ou recorrentes, pode ser útil reutilizar soluções previamente encontradas. Para isso, pode-se incrementar os algoritmos com componentes de memória, que funcionam como um local para armazenamento de soluções anteriores. A memória pode ser integrada de forma implícita, na forma de uma representação redundante no algoritmo ou explícita, como um componente separado. Essa abordagem é eficiente para

resolver problemas em ambientes cíclicos, além de favorecer a diversidade. No entanto, ela pode não ser útil quando o ambiente não retorna a estados semelhantes aos anteriores.

- **Abordagens preditivas.** Existem ambientes dinâmicos nos quais as mudanças apresentam padrões que se repetem e, portanto, podem ser previstos. Para esses casos, pode ser útil aprender os tipos de padrões e como se repetem, com a finalidade de predizer quando ocorrerão futuras mudanças. Existem várias técnicas para efetuar esse tipo de previsão, envolvendo

- **Métodos auto-adaptativos.** Para lidar com as mudanças no ambiente de busca, é possível usar mecanismos auto-adaptativos dos algoritmos evolutivos. Esses mecanismos, em geral, consistem em evoluir os próprios parâmetros da estratégia de mutação do algoritmo, baseado na aptidão da população.

- **Multi-populações.** O uso de multi-populações para resolver problemas de otimização dinâmica pode ser considerado como uma combinação das abordagens de manutenção de diversidade, uso de componentes de memória e de auto-adaptação. A ideia básica é manter mais de uma população, de modo que cada uma se encarregue de cuidar de uma determinada região do espaço de busca ou de rastrear as mudanças no ambiente.

2.3 Considerações Finais

A partir da fundamentação apresentada nas seções anteriores, podemos classificar o escalonamento dinâmico de projetos de software como um problema de otimização dinâmica multiobjetivo. Esta variante do SPSP, conhecida como DSPSP (do inglês, *Dynamic Software Project Scheduling Problem*), é objeto de estudo deste trabalho. As características de imprevisibilidade próprias da execução de projetos de software podem ser modeladas para se adequar ao uso de técnicas de otimização dinâmica.

Uma vez que uma necessidade de reescalonamento de um projeto produz a transformação do cronograma original em uma nova versão que, normalmente, apresenta considerável semelhança com a anterior. Dessa forma, entre as possíveis maneiras de tratar esses eventos dinâmicos estão a detecção de quais foram as mudanças ocorridas entre os dois cronogramas, a adição de componentes de memória ao algoritmo e a manutenção de diversidade no decorrer da busca. Essas abordagens podem ser combinadas entre si por meio do uso de múltiplas populações, cuja composição inicial de cada uma pode ser formada por soluções provenientes da aplicação de uma ou mais dessas técnicas. No capítulo seguinte, detalhamos a formulação do DSPSP e apresentamos uma proposta de abordagem baseada nos princípios da otimização dinâmica para sua resolução.

3

Algoritmos de Otimização Multiobjetivo Aplicados ao DSPSP

Neste capítulo serão apresentadas as técnicas utilizadas neste trabalho para resolver instâncias do DSPSP, bem como o algoritmo desenvolvido que incorpora características de múltiplos enxames. A seção 3.1 apresenta o *framework* desenvolvido para a execução dos experimentos, bem como seus principais componentes. Depois, a seção 3.2 explica como o DSPSP foi representado neste trabalho. Por fim, a seção 3.3 discorre sobre as meta-heurísticas de otimização utilizadas no desenvolvimento dos experimentos, incluindo a apresentação do funcionamento do algoritmo proposto.

3.1 *Framework* `spsp-jmetal`

Para avaliar o algoritmo proposto em relação às questões de pesquisa, foi desenvolvido um programa chamado `spsp-jmetal`. A finalidade do `spsp-jmetal` é servir como um *framework* de experimentos de meta-heurísticas populacionais aplicadas ao SPSP e DSPSP. A versão do `spsp-jmetal` com a qual este trabalho foi desenvolvido utiliza a versão 5.2 do *framework* jMetal (DURILLO; NEBRO, 2011) para implementar simulações de uso dos modelos do SPSP e DSPSP descritos nos capítulos anteriores. A escolha do jMetal deve-se à sua ampla utilização em estudos sobre meta-heurísticas aplicadas a problemas de otimização, uma vez que traz implementações suficientemente testadas dos principais algoritmos utilizados na área.

O `spsp-jmetal` implementa um modelo de DSPSP baseado no código MATLAB do experimento original de Shen et al. (2016), gentilmente disponibilizado pelos autores para a presente pesquisa. A implementação em Java utilizada neste trabalho busca construir o modelo de acordo com os *design patterns* predefinidos no jMetal 5.2. A interface `Problem` da biblioteca é implementada pela classe `DSPSProblem` do `spsp-jmetal`, permitindo que os algoritmos disponíveis no jMetal sejam aplicados ao DSPSP sem necessidade de ajustes adicionais.

Para facilitar as operações relacionadas à matriz de dedicação do DSPSP, as soluções são

encapsuladas pela classe `DedicationMatrix`, que codifica o vetor unidimensional de números reais `DoubleSolution` do jMetal em um objeto que encapsula uma matriz bidimensional de números reais e as operações necessárias para acessar os valores de dedicação de cada empregado a cada tarefa.

Uma das principais finalidade do `spsp-jmetal` é facilitar a implementação de experimentos nos quais diferentes algoritmos aplicados ao SPSP e suas variações são comparados entre si. Para isso, a classe `ExperimentRunner` recebe por meio de interface de linha de comando um arquivo no formato JSON com as informações sobre a parametrização do experimento, o que inclui uma lista de algoritmos a serem executados, quantas vezes esses algoritmos serão executados para cada instância, o número de avaliações da função objetivo e a uma lista com os arquivos de configuração de cada instância a ser avaliada. Os arquivos de configuração das instâncias, por sua vez, também são estão em formato JSON, tendo sido convertidos a partir dos originais utilizados em (SHEN et al., 2016) por meio de um *script* MATLAB também desenvolvido na presente pesquisa.

Depois que a configuração do experimento é carregada em uma instância da classe `ExperimentSettings`, as execuções dos algoritmos têm início. O fluxo principal de cada execução é controlado pela classe `ExperimentRunner`, que por sua vez utiliza a classe `AlgorithmAssembler` para construir as instâncias de execução de cada algoritmo a partir dos seus respectivos parâmetros de configuração.

As informações sobre tudo o que envolve o projeto sendo simulado são controladas e armazenadas em instâncias das classes `Project` e `DynamicProject` para SPSP e DSPSP, respectivamente. Entre as operações que essas classes disponibilizam estão as avaliações das funções objetivo de cada problema.

Cada evento dinâmico que ocorre na simulação de uma instância do projeto, dispara uma nova execução do algoritmo em experimentação no momento. Ao final de cada execução são gerados três arquivos: um com os vetores das soluções não-dominadas, outro com os valores das avaliações das funções objetivo para cada solução e outro com os valores normalizados das funções objetivo, para fins de cálculo das métricas de desempenho.

O `spsp-jmetal` conta com um mecanismo de *log* com *timestamp* e configurável quanto ao nível de severidade (*info*, *debug* ou *trace*). Esse recurso mostrou-se fundamental no caso do DSPSP pois, devido à complexidade do modelo e à grande quantidade de tempo necessária para rodar os experimentos, é preciso ter um meio de verificar o andamento da execução e, principalmente, as informações sobre a simulação do projeto em cada ponto de reescalonamento.

3.2 Representação do Problema

Esta seção detalha a modelagem para o DSPSP utilizada neste trabalho, que segue o modelo proposto por Shen et al. (2016), apresentado na seção 2.1.3.

3.2.1 Eventos Dinâmicos

A principal diferença do DSPSP em relação SPSP estático é a ocorrência de eventos dinâmicos em determinados instantes da execução do projeto. Neste trabalho, consideramos t_l ($l = 0, 1, 2, ...$) como um determinado instante em que algum tipo de evento dinâmico ocorre (incluindo o instante inicial t_0), de modo que l indica a ordem sequencial da ocorrência do evento no projeto. Tais eventos têm impacto direto no cronograma do projeto, pois interferem na composição da equipe e na alocação de seus integrantes para a execução das tarefas. Na construção do modelo utilizado neste trabalho, são considerados os seguintes eventos:

1. **Chegada de novas tarefas.** No decorrer do projeto é comum o surgimento de novos requisitos ou a modificação de requisitos já existentes. Com isso, novas tarefas podem ser incluídas no projeto a qualquer tempo, criando um ambiente em constante evolução. Uma tarefa pode ser classificada como urgente ou regular. Tarefas urgentes são aquelas que precisam ser executadas imediatamente, mesmo que para isso alguma tarefa em andamento precise ser interrompida. Tarefas regulares, por sua vez, podem ser agendadas para o momento adequado, respeitando as dependências entre as tarefas já existentes no cronograma.

2. **Saída de empregados.** Empregados podem precisar deixar o projeto temporariamente por motivo de doença ou de participação em outros projetos e atividades, entre outras razões. Nesses casos, o cronograma precisa ser adaptado para distribuir as tarefas em execução entre os empregados que continuam no projeto.

3. **Retorno de empregados.** As saídas de um empregado não são necessariamente definitivas. Portanto, um empregado que tenha saído do projeto pode retornar a qualquer tempo, ficando disponível para ser alocado nas tarefas em execução.

3.2.2 Incerteza na estimativa de esforço

Além dos eventos que ocorrem em determinados pontos no tempo (subseção 3.2.1), o modelo leva em consideração a incerteza inerente a qualquer estimativa de esforço de tarefas. Para isso, considera-se que o valor do esforço segue uma distribuição normal $\phi(\mu, \sigma)$, de modo que as tarefas possuem esforço estimado médio μ e desvio padrão σ. Dessa forma, no início do projeto assume-se o valor de μ para o esforço estimado. Cada vez que ocorre um evento dinâmico, aproveita-se o reescalonamento necessário para efetuar uma nova estimativa de esforço restante

para cada tarefa. O novo valor é obtido escolhendo aleatoriamente um valor da distribuição $\phi(\mu,\sigma)$ e, em seguida, subtraindo o esforço já realizado.

3.2.3 Propriedades dos empregados

Os membros da equipe de um projeto são chamados *empregados*. Assim como na formulação do SPSP estático, há um conjunto E de empregados que podem ser alocados a tarefas de um projeto e há conjunto S de habilidades requeridas para a execução de todas as tarefas do projeto.

Cada empregado e_i, sendo $i = 1,2,...,|E|$, possui alguns atributos que se mantêm fixos durante todo o projeto. No entanto, devido à possibilidade de um empregado e_i sair ou retornar ao projeto em um dado instante t_l, considera-se também um atributo binário $e_i^{disponivel}(t_l)$ que indica se e_i pode ou não se alocado a alguma tarefa no instante t_l. A Tabela 2 resume a descrição dos atributos dos empregados.

Tabela 2 – Propriedades dos empregados.

Atributo	Descrição
e_i^{hab}	Conjunto de valores que indica o quanto o empregado e_i domina cada uma das habilidades contidas em S. Os valores variam entre 0 e 5 e indicam a proficiência de e_i empregado na tarefa correspondente.
hab_i	Conjunto de habilidades nas quais e_i possui algum nível de proficiência.
e_i^{maxded}	Valor da dedicação máxima de e_i ao projeto. É um percentual do expediente normal de um empregado. $e_i^{maxded} = 1$ significa dedicação total. É possível atribuir tempo parcial ou permitir horas extras.
$e_i^{salario\_normal}$	Valor do salário mensal de e_i referente ao expediente regular.
$e_i^{salario\_ext}$	Valor do salário mensal de e_i referente a horas extras.
$e_i^{disponivel}(t_l)$	Variável binária que indica se e_i está disponível no instante t_l.
$e_{ij}^{proficiencia}$	Nível de proficiência de e_i na tarefa T_j.

Fonte: Próprio autor

3.2.4 Propriedades das tarefas

As atividades de projetos geralmente guardam algum grau de dependência entre si. Isso significa que, antes que uma determinada tarefa possa começar, muitas vezes há uma ou mais tarefas que devem ser concluídas. Essa característica permite que a relação de dependência entre as tarefas de um projeto possa ser modelada por meio de um grafo acíclico direcionado $G(V,A)$, onde V é o conjunto de vértices que representa as tarefas e A é o conjunto de arestas que representa o relacionamento de dependência entre duas tarefas (T_i,T_j). A Tabela 3 lista as descrições dos atributos de uma tarefa.

O valor de $t_j^{disponivel}(t_l) = 1$ se e somente se as três condições a seguir são verdadeiras: (1) T_j está incompleta, ou seja, ainda há esforço a ser realizado; (2) para qualquer habilidade

Tabela 3 – Propriedades das tarefas.

Atributo	**Descrição**
req_j	Conjunto de habilidades requeridas pela tarefa T_j.
$T_j^{esf\_tot\_est}$	Esforço total estimado no início do projeto para a execução da tarefa T_j. A incerteza desse valor segue a distribuição normal $\phi(\mu_j, \sigma_j)$.
$T_j^{incompleta}(t_l)$	Variável binária que indica se no instante t_l a tarefa T_j ainda está incompleta ($T_j^{incompleta}(t_l) = 1$) ou se já foi finalizada $T_j^{incompleta}(t_l) = 0$.
$T_j^{disponivel}(t_l)$	Variável binária que indica se no instante t_l a tarefa T_j está disponível ($T_j^{disponivel}(t_l) = 1$) ou não $T_j^{disponivel}(t_l) = 0$.
GPT	Grafo de Precedência de Tarefas. É um grafo acíclico direcionado $G(V,A)$ que representa os relacionamentos de dependência entre as tarefas. Em qualquer instante t_l da execução do projeto, o *GPT* pode ser atualizado para incluir novas tarefas ou remover as tarefas já concluídas.

Fonte: Próprio autor

requerida por T_j há pelo menos um empregado em t_l que possui esta habilidade e (3) todas as tarefas incompletas que precedem T_j no *GPT* satisfazem a condição anterior.

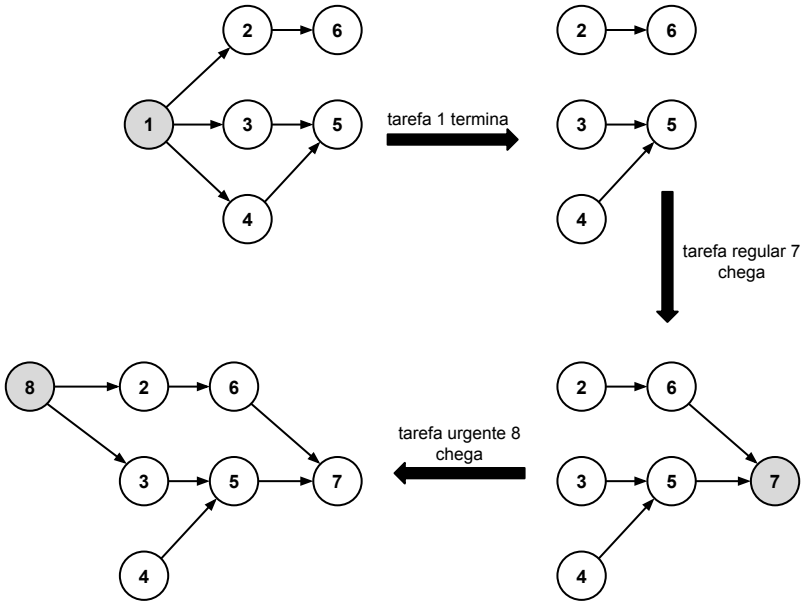
O processo de atualização do *GPT* consiste em remover os vértices das tarefas quando estas são concluídas e adicionar novos vértices quando novas tarefas chegam. Quando uma tarefa regular chega, ela é inserida após uma das tarefas incompletas. Caso a nova tarefa seja urgente, ela é inserida antes de uma ou mais tarefas incompletas, fazendo com que estas tenham seu processamento interrompido até que a nova tarefa urgente seja concluída.

A Figura 4 ilustra alguns exemplos de atualização do *GPT*. Quando a tarefa 1 é concluída, o processo de atualização a remove do grafo. Em seguida, chega a tarefa regular 7 é adicionada após as tarefas incompletas 5 e 6. Por fim, a tarefa urgente 8 chega e é adicionada antes das tarefas 2 e 3, que por sua vez são interrompidas porque passam a depender da finalização da tarefa 8.

3.2.5 Representação da solução

Seguindo a mesma definição de Alba e Chicano (2007) detalhada na seção 2.1.1, as soluções para o DSPSP são representadas na forma de uma matriz $X = x_{ij}$. A cada reescalonamento, busca-se encontrar um novo cronograma $X(t_l)$ no qual o valor de $x_{ij}(t_l)$ indica o percentual do expediente normal que um empregado e_i dedica à tarefa T_j no instante t_l. Uma vez que há eventos que afetam a disponibilidade de empregados, o valor de $x_{ij}(t_l) = 0$ para todo $j = 1,2,...,|T|$ se $e_i^{disponivel}(t_l) = 0$, bem como $x_{ij}(t_l) = 0$ para todo $i = 1,2,...,|E|$ quando $T_j^{disponivel}(t_l) = 0$. Sendo assim, somente os valores de $x_{ij}(t_l) \in \{x_{ij}(t_l) | e_i^{disponivel}(t_l) = 1 \wedge T_j^{disponivel}(t_l) = 1\}$ devem ser levados em consideração pelo método de otimização.

Figura 4 – Exemplo de atualização do Grafo de Precedência de Tarefas



Fonte: Próprio autor, adaptado de Shen et al. (2016).

3.2.6 Avaliação dos objetivos

A avaliação dos objetivos é feita de acordo com o modelo proposto por (SHEN et al., 2016) (seção 2.1.3) considerando que, em um dado instante t_l, pode-se obter as seguintes informações do projeto:

- O conjunto de empregados disponíveis $E\_disp(t_l)$;
- O conjunto de tarefas disponíveis $T\_disp(t_l)$;
- O GPT $G(V(t_l), A(t_l))$ atualizado.

Dessa forma é possível avaliar as funções-objetivo a serem otimizadas. Os objetivos estão sumarizados na Tabela 4.

Tabela 4 – Objetivos a serem otimizados.

Função	**Objetivo**	**Descrição**
$f_1(t_l)$	Duração	Tempo decorrido entre o instante t_l e o encerramento do projeto.
$f_2(t_l)$	Custo	Valor a ser pago aos empregados entre o instante t_l e o encerramento do projeto.
$f_3(t_l)$	Robustez	Medida do quanto o cronograma é suscetível a incertezas na estimativa de esforço. Quanto menor, melhor o desempenho.
$f_4(t_l)$	Estabilidade	Medida do quanto o cronograma em t_l difere do cronograma anterior.

Fonte: Próprio autor

A *duração* representa o tempo necessário para que todas as tarefas restantes no instante t_l sejam concluídas com os recursos disponíveis no momento.

$$f_1(t_l) = duracao_I = \max_{\{j|T_j \in T\_disp(t_l)\}} (T_j^{fim}(t_l)) - \min_{\{j|T_j \in T\_disp(t_l)\}} (T_j^{inicio}(t_l)) \tag{3.1}$$

Na equação 3.1, o subscrito I refere-se ao cenário inicial, no qual considera o esforço estimado restante $T_j^{esf\_tot\_est}$ o esforço necessário para completar a tarefa T_j a partir de um dado instante t_l. Para cada uma das tarefas disponíveis T_j em t_l, é considerado apenas o esforço restante, descartando o esforço já realizado até o momento. $T_j^{inicio}(t_l)$ é o ponto no tempo, em meses, no qual uma tarefa T_j começa a ser executada de acordo com o novo cronograma vigente. $T_j^{fim}(t_l)$, por sua vez, refere-se ao momento em que uma tarefa T_j é concluída.

O *custo* associado a um cronograma é o total de despesas com empregados de acordo com a dedicação deles às tarefas disponíveis no instante t_l.

$$f_2(t_l) = custo_I = \sum_{t' \geq t_l} \sum_{e_i \in e\_disp(t_l)} e\_custo_i^{t'} \tag{3.2}$$

Na equação 3.2, $e\_custo_i^{t'}$ é o valor pago ao empregado e_i no instante t'. Considerando a dedicação total de e_i em t', esse custo é obtido pela Equação 3.3 quando $\sum_{j \in T\_ativas(t')} x_{ij}(t_l) \leq 1$ e pela Equação 3.4 quando $1 < \sum_{j \in T\_ativas(t')} x_{ij}(t_l) \leq e_i^{maxded}$.

$$e\_custo_i^{t'} = e_i^{salario\_normal} \cdot t' \cdot \sum_{j \in T\_ativas(t')} x_{ij}(t_l) \tag{3.3}$$

$$e\_custo_i^{t'} = e_i^{salario\_normal} \cdot t' \cdot 1 + e_i^{salario\_extra} \cdot t' \cdot (\sum_{j \in T\_ativas(t')} x_{ij}(t_l) - 1) \tag{3.4}$$

Para obter a *robustez* da solução é usado um método baseado em cenários, que consiste em gerar um conjunto $\{\theta_q | q = 1, 2, \cdots, N\}$ de cenários aleatórios de duração e custo para as tarefas do projeto. Esses cenários são utilizados para o cálculo da robustez do cronograma em um dado tempo t_l, conforme (3.5), onde $duracao_I$ e $custo_I$ são respectivamente a duração e o custo iniciais do projeto e λ é um parâmetro de peso que determina a importância relativa do custo sobre a duração.

$$\begin{aligned} f_3(t_l) = robustez = & \sqrt{\frac{1}{N}\sum_{q=1}^{N}\left(max\left(0, \frac{duracao_q(t_l) - duracao_I(t_l)}{duracao_I(t_l)}\right)\right)^2} \\ & + \lambda\sqrt{\frac{1}{N}\sum_{q=1}^{N}\left(max\left(0, \frac{custo_q(t_l) - custo_I(t_l)}{custo_I(t_l)}\right)\right)^2} \end{aligned} \tag{3.5}$$

A avaliação da *estabilidade* em um tempo t_l é feita por meio da soma ponderada das diferenças entre as dedicações de cada empregado a cada tarefa restante, em relação ao tempo t_{l-1}. Os pesos (ω_{ij}) servem para penalizar cronogramas nos quais empregados são realocados para uma nova tarefa, simulando o tempo adicional necessário para se familiarizar com ela (equação 3.6).

$$f_4(t_l) = estabilidade = \sum_{\{i|e_i \in e\_disp(t_{l-1}) \cap e\_disp(t_l)\}} \sum_{\{j|T_j \in T\_disp(t_{l-1}) \cap T\_disp(t_l)\}} \omega_{ij} |x_{ij}(t_l) - x_{ij}(t_{l-1})| \tag{3.6}$$

onde e_i é um empregado, T_j é uma tarefa, ω_{ij} é um elemento da matriz de pesos, x_{ij} é um elemento da matriz de dedicações e $e\_disp(t_l)$ e $T\_disp(t_l)$ são respectivamente os conjuntos de empregados e tarefas disponíveis num dado tempo t_l.

3.2.7 Restrições

Para que uma solução do DSPSP seja considerada válida, as três restrições a seguir devem ser respeitadas.

R1. Carga de trabalho

Em um instante t' após o ponto de reescalonamento t_l a dedicação total de um empregado disponível a todas as tarefas ativas no momento não deve exceder a sua dedicação máxima ao projeto (Equação 3.7). Por exemplo, caso $e_i^{maxded} = 1.2$, então é permitido a e_i trabalhar horas extras, desde que não excedam 20% do expediente normal.

$$\begin{aligned} &\forall e_i \in e\_disp(t_l), \forall t' \geq t_l, \text{ onde} \\ &e\_carga_i^{t'} = \sum_{j \in T\_ativas(t')} x_{ij}(t_l), \text{e } e\_carga_i^{t'} \leq e_i^{maxded} \end{aligned} \tag{3.7}$$

R2. Habilidades dos empregados

Todos os empregados disponíveis que trabalham juntos em uma determinada tarefa disponível devem, coletivamente, abranger todas as habilidades requeridas para a tarefa em questão. Simultaneamente, cada tarefa disponível em t_l deve ser executada por pelo menos um empregado disponível. Esta restrição é formalizada de acordo com a Equação 3.8.

$$\begin{aligned} &\forall T_j \in T\_disp(t_l) \\ &\text{onde } hab_j \subseteq \cup_{e_i \in e\_disp(t_l)} \{hab_i | x_{ij}(t_l) > 0\} \end{aligned} \tag{3.8}$$

R3. Limite de empregados por tarefa

O número de empregados disponíveis trabalhando juntos em uma tarefa T_j não deve exceder o limite superior $T_j^{max\_emp}$ definido na instância do projeto. No ponto de reescalonamento t_L, chamamos de $T_j^{tam\_time}(t_l)$ o tamanho da equipe alocada para a tarefa T_J e de $T_j^{num\_min\_emp}(t_l)$ o número mínimo de empregados disponíveis que devem atuar em T_j para satisfazer a restrição R2. Dessa forma, a restrição pode ser descrita de acordo com a Equação 3.9.

$$\forall T_j \in T\_disp(t_l), T_j^{tam\_time}(t_l) \leq \max\left(T_j^{max\_emp}, T_j^{num\_min\_emp}(t_l)\right) \tag{3.9}$$

Ao contrário das restrições anteriores R1 e R2, esta restrição R3 pode ser relaxada mediante penalização descrita em 3.2.8, caso não seja satisfeita.

3.2.8 Tratamento das restrições

Nos casos em que as soluções encontradas violam alguma das restrições, é preciso tratá-las para que sua avaliação não interfira na busca por soluções válidas. No modelo DSPSP proposto, as restrições são tratadas conforme segue:

- **R1. Carga de trabalho.** Quando uma restrição é violada, o ideal é que o valor da função objetivo para a solução que está sendo avaliada sofra uma penalização. No entanto, devido à dificuldade em satisfazer a restrição de carga de trabalho, a solução é reparada por meio de normalização, de modo a acomodar todas as tarefas às quais os empregados estão alocados. Em um instante t', caso a restrição de carga de trabalho para um empregado e_i seja violada, ou seja, $e\_carga_i^{t'} > e_i^{maxded}$, então sua dedicação $x_{ij}(t_l)$ para cada tarefa T_j que está sendo executada em t' é dividida por $e\_carga_i^{t'}$. Para possibilitar comparação, o valor normalizado da dedicação $x_{ij}(t_l)$ é denotado por $d_{ij}(t_l)$, de modo que $d_{ij}(t_l) = x_{ij}(t_l)/\max 1, e\_carga_i^{t'}/e_i^{maxded}$. Se $e\_carga_i^{t'} \leq e_i^{maxded}$, então a dedicação não é normalizada.

- **R2. Habilidades dos empregados.** Para tratar esta restrição, leva-se em consideração a proficiência de cada empregado nas habilidades requeridas pelas tarefas. Para isso, é preciso saber qual a dedicação total a uma tarefa $T_j(t_l)$. Então, inicialmente obtém-se a dedicação total Td_j de todos os funcionários disponíveis:

$$Td_j = \sum_{e_i \in e\_disp(t_l)} d_{ij}(t_l) \tag{3.10}$$

A partir desse valor, calcula-se um valor que indica qual a aptidão total (*fitness*) de todos os empregados para a tarefa $T_j(t_l)$:

$$F_j(t_l) = \left(\sum_{e_i \in e\_disp(t_l)} e_{ij}^{proficiencia} \cdot d_{ij}(t_l) \right) / Td_j(t_l), \tag{3.11}$$

onde $F_j(t_l)$, na verdade, é uma fração da dedicação total dos empregados à tarefa $T_j(t_l)$, uma vez que é levada em conta a proficiência de cada empregado nas habilidades requeridas pela tarefa. Nos casos em que a proficiência não seja total, o valor da aptidão é reduzido, o que acarreta em maior tempo para a execução da tarefa. Em seguida, o valor de $F_j(t_l)$ é convertido no custo unitário $V_j(t_l)$:

$$V_j(t_l) = \max(1, 8 - round(F_j(t_l) * 7 + 0.5)) \tag{3.12}$$

O custo unitário varia no intervalo [1, 7], sendo que quanto menor o valor, mais os empregados alocados para a tarefa T_j são aptos a executá-la. Por fim, obtém-se a dedicação total ajustada $A\_Td_j(t_l)$ de acordo com a aptidão dos empregados:

$$A\_Td_j(t_l) = Td_j(t_l)/V_j(t_l) \tag{3.13}$$

Uma vez conhecida a dedicação total ajustada e sendo $T_j^{esf\_rest}(t_l)$ o esforço restante para concluir T_j, é possível obter o tempo t_j^{rest} necessário para concluí-la:

$$t_j^{rest} = T_j^{esf\_rest}/A\_Td_j(t_l) \tag{3.14}$$

Caso um cronograma candidato em t_l seja inviável porque as habilidades de algumas tarefas não são cobertas pelos empregados disponíveis, então valores elevados de penalidades são atribuídos ao valor das funções objetivo. Sendo *reqsk* o número de habilidades faltantes, os objetivos f_1, f_2, f_3 e f_4 são penalizados de acordo com as equações 3.15, 3.16, 3.17 e 3.18, respectivamente.

$$f_1(t_l) = duracao_I = reqsk \cdot 14k \cdot \sum_{T_j \in T\_disp(t_l)} T_j^{esf\_tot\_est}(t_l) / \min_{e_i \in e\_disp(t_l)} e_i^{maxded} \tag{3.15}$$

$$f_2(t_l) = custo_I = reqsk \cdot 14 \cdot \sum_{e_i \in e\_disp(t_l)} \sum_{T_j \in T\_disp(t_l)} e_i^{salario\_extra} \cdot T_j^{esf\_tot\_est}(t_l) \cdot 7 \tag{3.16}$$

$$f_3(t_l) = robustez = reqsk \cdot 2 \cdot C_{rob} \tag{3.17}$$

$$f_4(t_l) = estabilidade = reqsk \cdot 2 \cdot |e\_disp(t_l)| \cdot |T\_disp(t_l)| \cdot \max_{e_i \in e\_disp(t_l)} e_i^{maxded}, \quad (3.18)$$

onde C_{rob} e k são parâmetros constantes que, neste trabalho, assumem os valores 100 e 1, respectivamente. Tais constantes, assim como os demais valores fixos observados nas equações, visam garantir que as quatro penalizações acima retornem valores maiores que os valores dos objetivos de qualquer solução viável válida. Os detalhes do desenvolvimento de cada equação, assim como a motivação para o uso de determinados valores fixos, são demonstrados em (SHEN et al., 2016).

- **R3. Limite de empregados por tarefa.** O tratamento dos casos em que o número de empregados alocados a uma tarefa excede o limite é feito reparando a solução em duas etapas. Na primeira, atribui-se $x_{ij}(t_l) = 0$ à dedicação do empregado e_i à tarefa T_j caso este não possua nenhuma das habilidades requeridas pela tarefa. Em seguida, verifica-se se o tamanho da equipe de cada tarefa disponível $T_j \in T\_disp(t_l)$ é, maior que o limite de empregados permitido $T_j^{max\_emp}$. Se esse limite é excedido $T_j^{max\_emp}$, o seguinte procedimento é executado: 1) ordenar os empregados da equipe da tarefa T_j de acordo com sua proficiência $e_{ij}^{proficiencia}$; 2) iniciando do empregado com a menor proficiência, verificar se pode ser removido da equipe sem violar a restrição R2 e removê-lo se for o caso; 3) repetir o passo anterior para os demais empregados do time até que todos tenham sido verificados. Caso o tamanho da equipe não possa ser reduzido a $T_j^{max\_emp}$ sem violar R2, uma penalidade é aplicada ao esforço da tarefa T_j, como forma de modelar o *overhead* de comunicação necessário para suprir as lacunas de conhecimento da equipe. Para tanto, o cálculo da penalidade sobre o esforço $T_j^{esforco}$ quando $T_j^{tam\_time} > T_j^{max\_emp}$ é calculado conforme a Equação 3.19

$$T_j^{esforco} = T_j^{esforco} \cdot \left(1 + \frac{T_j^{tam\_time} \cdot \left(T_j^{tam\_time} - 1\right)/2}{Z}\right), \quad (3.19)$$

onde Z é um parâmetro, que regula a relação entre o tempo para completar uma tarefa e o número de empregados.

3.2.9 Decisão sobre escolha da solução

Ao final de cada reescalonamento é preciso simular a tomada de decisão do gerente de projeto acerca da escolha do cronograma a ser adotado para o restante do projeto. O procedimento de reescalonamento gera uma população de soluções não-dominadas, dentre as quais é necessário decidir se será dada preferência a uma solução que tende a minimizar algum dos objetivos avaliados ou a equilibrá-los.

O modelo adotado neste projeto segue o que foi desenvolvido por Shen et al. (2016) e consiste nos passos descritos abaixo:

1. **Construção de uma matriz de comparação par-a-par.** A comparação entre os quatro objetivos a serem avaliados visa responder ao questionamento sobre qual deles é mais importante em relação ao demais. Sendo $N\_o = 4$ o número de objetivos, compara-se o objetivo f_i em relação a f_j, sendo $(i, j = 1, 2, ..., N\_o, j > i)$. Dessa forma, são feitas $N\_o \cdot (N\_o - 1)/2 = 4(4-1) = 6$ comparações para a tomada de decisão. A partir daí, pode ser construída uma matriz de comparação $C_1 = (c_{ij})_{N\_o \times N\_o}$ por meio da escala de nove pontos descrita pelo Processo Analítico de Hierarquia (do inglês *Analytic Hierarchy Process - AHP*) (FÜLÖP, 2005).

2. **Estimativa de um vetor de pesos para os múltiplos objetivos.** O segundo passo é estimar um vetor de pesos $w = (w_i)_{N\_o \times 1}$ por meio do método de mínimos quadrados logarítmico (SAATY; VARGAS, 1984). A média geométrica de cada linha de C_1 é calculada e depois normalizada.

3. **Normalização dos valores dos objetivos.** Cada um dos objetivos é normalizado de acordo com a Equação 3.20, onde f_i^{max} e f_i^{min} são os valores máximo e mínimo dos objetivos entre todas as soluções não-dominadas obtidas no ponto de escalonamento atual.

$$n\_f_i(x) = (f_i^{max} - f_i(x))/(f_i^{max} - f_i^{min}), i = 1, 2, ..., N\_o \tag{3.20}$$

4. **Cálculo do valor de utilidade.** A média geométrica ponderada dos múltiplos valores dos objetivos é usada para obter o valor de utilidade de cada solução não-dominada, conforme a Equação 3.21.

$$U(x) = \prod_{i=1}^{N\_o} n\_f_i(x)^{w_i / \Sigma_i^{N\_o} w_i} \tag{3.21}$$

5. **Escolha da solução com maior valor de utilidade.** Uma vez calculados os valores de utilidade para todas as soluções não-dominadas no ponto de reescalonamento atual, a solução com que possui o maior valor de utilidade é escolhida como a que será adotada como cronograma vigente.

3.3 Algoritmos de Otimização Multiobjetivo Aplicados ao DSPSP

Nesta seção apresentamos as meta-heurísticas e algoritmos utilizados nos experimentos deste trabalho. As implementações dos algoritmos são as fornecidas pelo *framework* jMetal, às quais foram incorporados recursos para suportar a reinicialização customizada de populações e o uso de múltiplos enxames.

3.3.1 Non-dominated Sorting Genetic Algorithm II

O *Non-dominated Sorting Genetic Algorithm II*, ou NSGA-II, é um algoritmo clássico de literatura sobre otimização multiobjetivo. Neste trabalho, o NSGA-II é utilizado para representar a meta-heurística de algoritmos genéticos na comparação com as meta-heurísticas baseadas em enxames de partículas.

Os algoritmos genéticos inspiram-se em eventos biológicos da área da genética, no qual as soluções para um problema de otimização são representadas por genes. Esses genes sofrem alterações provocadas por mutações e cruzamentos com outros genes. A ideia é que apenas os indivíduos mais adaptáveis sejam selecionados para perpetuar suas características nas gerações (iterações) seguintes.

A primeira versão do NSGA foi proposta em (SRINIVAS; DEB, 1994) e trazia a ideia de ordenar as soluções não-dominadas e basear seu operador de seleção nesse *ranking*. A segunda versão, o NSGA-II, foi proposta por (DEB et al., 2002) e obteve grande popularidade, inclusive sendo utilizada como fundamentação para outras meta-heurísticas (COELLO et al., 2007). No NSGA-II, os *rankings* dão origem a descendentes através da aplicação de operadores de mutação e cruzamento. Os melhores indivíduos entre esses descendentes do *ranking* são selecionados levando em conta a distancia de Crowding, como forma de manter a diversidade das soluções.

3.3.2 Particle Swarm Optimization

O algoritmo proposto na seção 3.3.3, baseia-se na combinação de duas ou mais instâncias de execução do algoritmo SMPSO (do inglês, *Speed-constrained Multiobjective Particle Swarm Optimization*) (NEBRO et al., 2008) que, por sua vez, é uma variante do algoritmo PSO (do inglês, *Particle Swarm Optimization*) (KENNEDY, 1995). A otimização por enxames de partículas é uma meta-heurística que na qual os indivíduos da população (*enxame*) do algoritmo são chamados de *partículas* e cada uma delas representa uma solução possível para o problema.

Partindo do princípio que as partículas se movimentam no espaço de estados, então cada uma delas possui atributos de *posição* e *velocidade*. A posição de uma partícula corresponde ao valor da solução que ela representa, enquanto sua velocidade é um vetor que determina a direção na qual uma partícula deve se mover para que a avaliação da função-objetivo para a sua posição atual melhore. A interação de cada partícula com as demais obedece a um conjunto de regras simples, de modo que a sequência de interações emula um comportamento inteligente. Esse comportamento baseia-se no referencial de posições das partículas com melhor avaliação da função objetivo.

Para tornar esse comportamento possível, o algoritmo precisa manter algumas informações sobre a busca, tais como: a melhor posição encontrada por cada partícula do enxame (*pbest*); a melhor posição que um vizinho da partícula atingiu até o momento (*lbest*); e a posição da melhor partícula de toda a população (*gbest*). As partículas que são usadas para guiar outras

partículas em direção a melhores regiões do espaço de busca são chamadas de *líderes*.

A velocidade das partículas é controlada por um fator chamado *peso inercial* (W), que controla o impacto do histórico de velocidades de uma partícula em sua velocidade atual. Outro parâmetro considerado no PSO é o *fator de aprendizado*, que representa a atração que uma partícula possui em relação a seu próprio sucesso ou de seus vizinhos. São utilizados dois fatores de aprendizado: o primeiro é o *coginitivo* (C_1), que representa a atração da partícula a seu próprio sucesso; e o segundo é o *social* (C_2), que indica a atração de uma partícula em direção ao sucesso dos seus vizinhos. Tanto C_1 quanto C_2 são definidas como constantes.

O conceito de vizinhança de partículas é determinado pela *topologia da vizinhança*, que representa o conjunto de partículas que contribuem para calcular o valor de *lbest* para uma determinada partícula. Essa conexão entre as partículas é representada como um grafo que, por sua vez, possui uma determinada topologia. Os tipos mais comuns de topologias são: Vazia (partículas isoladas); Estrela, Árvore, Grafo Completo e Melhor Local (Coello Coello; REYES-SIERRA, 2006).

A listagem do Algoritmo 1 apresenta o pseudocódigo do funcionamento geral do PSO.

Algoritmo 1: Pseudocódigo do algoritmo PSO genérico

```
1  Inicialize o enxame;
2  Localize o líder;
3  g = 0;
4  enquanto g < gmax faça
5      para Cada partícula p do enxame faça
6          Atualize a posição de p;
7          Avalie a função objetivo para p;
8          Atualize a melhor posição da partícula até o momento (pbest);
9      fim
10     Atualize o líder;
11 fim
```

3.3.3 Algoritmo Proposto (MS2MO)

Neste trabalho apresentamos um algoritmo de otimização baseado em múltiplos enxames de partículas, chamado *Multi-Swarm Multi-Strategy Algorithm for Many-Objective Optimization* (MS2MO). O método baseia-se no trabalho de Matos e Britto (2017), caracterizado pelo uso de múltiplos enxames para resolver problemas do tipo MaOP por meio de comunicação mútua periódica entre as populações. O uso de múltiplas populações tem apresentado bons resultados em problemas com muitos objetivos, além de ser uma maneira de garantir diversidade, característica útil no contexto da otimização dinâmica.

No MS2MO, cada um dos diferentes enxames que o compõem têm sua própria população. Durante a busca, essas populações compartilham informação indiretamente entre si por meio de

um arquivo externo. O arquivo externo é atualizado periodicamente com as melhores soluções encontradas por cada enxame. No momento da comunicação entre os enxames também há compartilhamento direto de soluções entre eles. Cada enxame possui seu próprio arquivo interno, que somente ele pode alterar. Quando a comunicação ocorre, as soluções de um enxame são passadas para outros, de acordo com a topologia utilizada. Os enxames receptores, por sua vez, não apagam os seus arquivos. Ao obter as informações compartilhadas, os receptores apenas tentam atualizar seus respectivos arquivos com as novas soluções. Os enxames podem utilizar diferentes métodos de arquivamento, possibilitando influência mútua.

Cada um dos enxames, por sua vez, executa um algoritmo MOPSO independente. Nesta abordagem, foram utilizadas instâncias do algoritmo SMPSO (NEBRO et al., 2009) adaptadas para as características de otimização dinâmica. A principal adaptação é a possibilidade de passar como parâmetro um conjunto de soluções pré-determinado para compor a população inicial. Dessa forma, é possível fazer com que cada enxame utilize uma ou mais estratégias heurísticas dinâmicas para melhorar a convergência. As estratégias em questão serão detalhadas mais adiante, na seção 3.3.3.3.

A comunicação entre os enxames segue a topologia de anel, na qual cada enxame possui dois vizinhos. Isso significa que os enxames conectam-se entre si de forma indireta, visto que um enxame i troca informações com os enxames $i-1$ e $i+1$ unicamente a cada momento de comunicação. A troca de informações entre os enxames consiste no envio das das soluções líderes para seus dois vizinhos e na atualização do seu arquivo apenas pelas soluções líderes vindas desses mesmos dois vizinhos.

Os métodos de arquivamento utilizados no MS2MO podem ser diferentes para cada enxame, de acordo com parametrização. Por motivos de simplificação da análise dos resultados, os experimentos realizados para este trabalho utilizam somente o método de *Crowding Distance* (NEBRO et al., 2008), que é o mesmo adotado pelo SMPSO.

A listagem do Algoritmo 2 resume em pseudocódigo o funcionamento do MS2MO.

3.3.3.1 Parâmetros do algoritmo

O MS2MO pode ser parametrizado de forma que seja possível experimentar diferentes combinações de características do algoritmo, na busca por melhores soluções para MaOPs dinâmicos. Neste trabalho experimental, as diferentes execuções do algoritmo foram realizadas com a possibilidade de variar a) a quantidade de avaliações da função objetivo, b) o número de enxames, c) o tamanho da população de cada enxame, d) as estratégicas heurísticas dinâmicas para reinicialização das populações (ver seção 3.3.3.3) e e) a proporção da população que é inicializada por cada estratégia. Entre outros parâmetros que não foram explorados neste trabalho mas também poderiam ser configurados estão o momento de comunicação entre as populações e os métodos de arquivamento utilizados pelos enxames.

Algoritmo 2: Pseudocódigo do algoritmo MS2MO

Entrada: Número de avaliações da função objetivo; Número de enxames; Número de partículas por enxame; Momento de comunicação entre os enxames
Saída: Conjunto de soluções não-dominadas entre todos os enxames

```
1  para Cada enxame s faça
2      Inicialize o enxame s;
3      para Cada partícula p do enxame s faça
4          Avalie a função objetivo para p;
5      fim
6  fim
7  Inicialize os líderes no arquivo externo;
8  enquanto Limite do número de iterações não atingido faça
9      para Cada enxame s faça
10         para Cada partícula p do enxame s faça
11             Selecione o líder do enxame;
12             Atualize a posição de p;
13             Realize a mutação de p;
14             Avalie a função objetivo para p;
15             Atualize a melhor posição da partícula até o momento (pbest);
16         fim
17         Atualize arquivo externo com os líderes do enxame;
18     fim
19     se Momento de comunicação entre os enxames então
20         Realiza troca de informação sobre os líderes entre os enxames vizinhos;
21     fim
22 fim
```

3.3.3.2 MS2MO aplicado ao DSPSP

Uma vez que o DSPSP caracteriza-se por ser um problema de otimização dinâmica com muitos objetivos, é possível utilizar o MS2MO para resolvê-lo. Para isso, foi desenvolvida uma adaptação do modelo que simula o ambiente de execução de um projeto de software. Nessa adaptação, as características de uma instância de projeto são passadas como parâmetro de entrada na simulação. Conforme detalhado na Seção 3.2, essas características incluem: a quantidade de empregados e suas habilidades; a quantidade de tarefas e as habilidades requeridas; e a sequência de diferentes eventos dinâmicos que ocorrem ao longo do projeto, incluindo tipo e instante de ocorrência do evento.

A simulação de execução de um projeto inicia com um conjunto de empregados que precisam ser alocados a um conjunto de tarefas. Todavia, antes do início do projeto, é preciso montar um cronograma que minimize sua duração e o custo, ao mesmo tempo em que garanta que ele seja robusto o suficiente quanto a incertezas provenientes das estimativa de esforço. Para isso, é realizado o *escalonamento inicial*, que consiste em uma primeira execução do MS2MO no instante inicial t_0. Nessa primeira execução, apenas três objetivos são avaliados: duração,

custo e robustez. A estabilidade não é avaliada no início pois seu cálculo depende da existência de cronogramas anteriores.

A primeira execução do MS2MO gera um conjunto de soluções não-dominadas que representam possíveis cronogramas otimizados a serem adotados no início do projeto. A decisão sobre qual das soluções será selecionada é tomada conforme procedimento descrito na seção 3.2.9, que retorna o *cronograma inicial*.

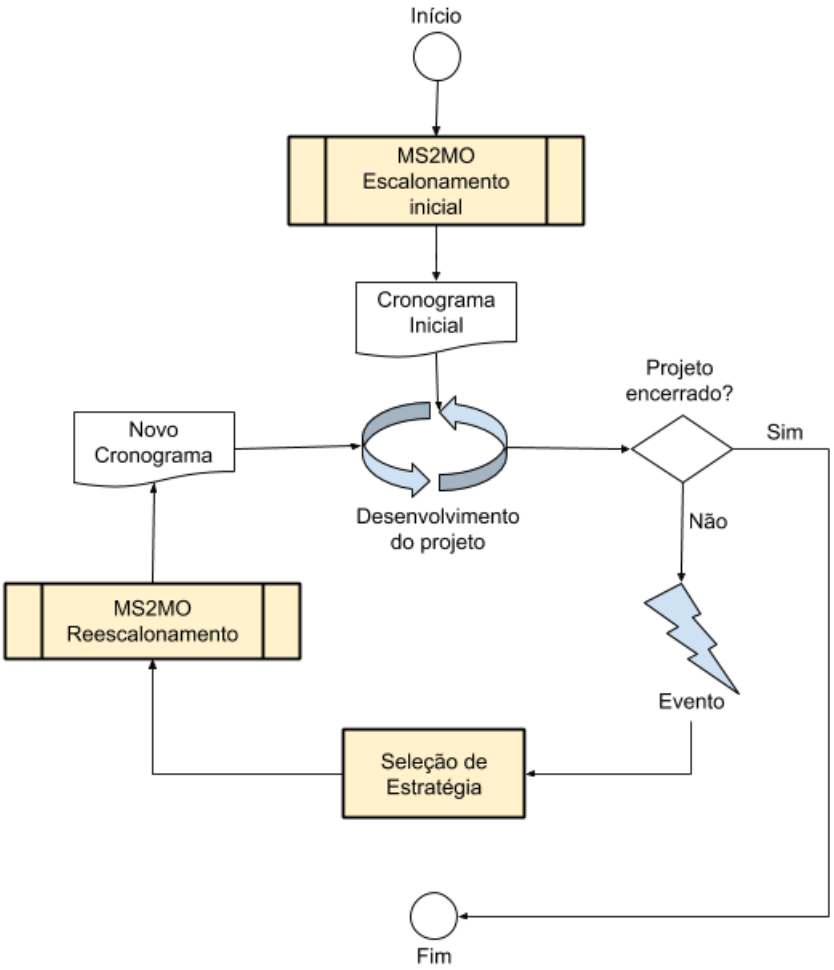
Uma vez definido o cronograma inicial, tem início o *desenvolvimento do projeto*, que se encerra quanto todas as suas tarefas tenham sido concluídas. No entanto, a execução do projeto está sujeita à sequência de *eventos dinâmicos* passada como parâmetro nas configurações da instância. Portanto, a depender do tipo do evento ocorrido, o MS2MO seleciona uma estratégia para ajustar o cronograma vigente, conforme sugerem Shen et al. (2016). Nesse ponto, cabe ressaltar que uma simulação de projeto é, na verdade, composta por várias execuções diferentes do algoritmo escolhido para resolver o problema.

No caso de um evento de saída de empregado, o ajuste consiste em remover do cronograma as alocações do ex-empregado e, em seguida, encontrar na equipe do projeto outros empregados que tenham alguma proficiência nas habilidades requeridas pelas tarefas que o ex-empregado vinha executando. Para o evento de retorno de um empregado ao projeto, o algoritmo atribui um valor mínimo para a dedicação do empregado nas tarefas em cujas habilidades requeridas ele tem alguma proficiência, respeitando a restrição de tamanho máximo da equipe da tarefa. Por meio de mutação da solução, o cronograma ajustado serve de base para a criação de um conjunto de indivíduos que possuem características semelhantes às da solução desejada.

A ocorrência de um evento em um dado instante t_l traz a necessidade de um *reescalonamento*. Os reescalonamentos consistem em novas execuções do MS2MO, sendo que aplicados aos empregados e tarefas disponíveis em t_l. Nesses casos, a inicialização das populações é feita através das estratégias heurísticas dinâmicas descritas na Seção 3.3.3.3. O conjunto de indivíduos gerados no passo anterior é então utilizado para compor parte dessa população inicial. Uma vez executado o algoritmo, as soluções geradas passam pelo processo de tomada de decisão, no qual é escolhido o novo cronograma a ser seguido a partir de então.

O desenvolvimento do projeto então prossegue, sujeito aos eventos dinâmicos da sequência definida na instância — e seus respectivos reescalonamentos —, até que todas as suas tarefas tenham sido finalizadas. Vale ressaltar que é possível adaptar outros algoritmos além do MS2MO para a execução da simulação desse modelo, conforme demonstrado nos experimentos deste trabalho. O fluxo de execução de uma simulação de projeto usando MS2MO é ilustrado pela Figura 5.

Figura 5 – Fluxo de execução da simulação de projetos com MS2MO



Fonte: Próprio autor.

3.3.3.3 Estratégias heurísticas dinâmicas

A exemplo do trabalho de Shen et al. (2016), o algoritmo proposto explora as características dinâmicas do DSPSP para obter melhor desempenho na otimização. No DSPSP, cada evento ocorrido no projeto marca um ponto de reescalonamento. A depender do evento ocorrido uma determinada estratégia é escolhida (ver Figura 5, *Seleção de Estratégia*) e é disparada uma nova execução do algoritmo (ver Figura 5, *MS2MO Reescalonamento*). As soluções resultantes são armazenadas em um histórico e a solução escolhida pelo processo de decisão passa a ser o cronograma vigente.

As estratégias heurísticas dinâmicas consistem em construir a população inicial da busca utilizando-se desse conhecimento armazenado. Parte-se do princípio que soluções bem avaliadas em um instante anterior podem estar bem próximas das soluções desejadas para o instante atual. Dessa forma, é possível reparar soluções históricas — bem como o cronograma vigente — de acordo com as características do evento dinâmico ocorrido. Quando um ponto de reescalonamento

é atingido, essas soluções reparadas podem ser usadas para compor a população inicial de cada nova execução do algoritmo. Neste trabalho os algoritmos avaliados têm a possibilidade de incorporar três estratégias heurísticas para iniciar suas populações, conforme detalhado a seguir.

Uso de informações históricas

Supondo que cada evento dinâmico introduz poucas mudanças no espaço de busca, pode-se aproveitar as alocações de dedicação dos empregados obtidas pelo processo de reescalonamento anterior. Neste algoritmo, isso é feito por meio da incorporação de algumas das soluções não-dominadas encontradas pelo execução no instante t_{l-1}.

Ajuste Proativo

Uma vez que cada evento dinâmico consiste na informação do que mudou no espaço de busca, é possível antecipar-se a essa mudança promovendo ajustes na solução corrente, ou seja, no cronograma vigente até o instante t_l. Um recurso semelhante é usado por Shen et al. (2016) para criar a população inicial. Neste trabalho, o ajuste proativo consiste em utilizar o cronograma vigente como base para a criação de uma solução reparada que atenda ao novo estado do espaço de busca. Uma vez efetuado o reparo, este novo cronograma é incluído no enxame inicial de partículas, juntamente com outras soluções derivadas dele por meio de mutação.

O ajuste é aplicado quando ocorrem eventos de *saída* ou de *retorno* de empregados pois durante a execução dos experimentos, estes tipos de evento foram os que provocaram maior disrupção no processo de busca. Um exemplo são os casos em que a saída de um empregado faz com que não haja mais nenhum membro remanescente na equipe com as habilidades necessárias para executar uma ou mais tarefas ativas no momento.

Introdução de partículas aleatórias

Com o intuito de garantir alguma diversidade às soluções, o algoritmo proposto sempre constrói os enxames contendo uma certa quantidade de soluções aleatórias. Essa quantidade varia de acordo com a parametrização da proporção de soluções históricas e da eventual inclusão de soluções criadas por meio do ajuste proativo do cronograma vigente.

4 Experimentos

Este capítulo descreve os experimentos realizados para responder às questões de pesquisa que norteiam este trabalho, investigando como o algoritmo proposto e a incorporação de técnicas de otimização dinâmica podem ser aplicadas ao problema, bem como comparando o MS2MO a outros algoritmos da literatura. Na Seção 4.1 é detalhada a metodologia utilizada na condução do experimento. A Seção 4.2 traz os resultados experimentais da validação do algoritmo. Por fim, a Seção 4.3 discorre sobre algumas das possíveis ameaças à validade deste estudo.

4.1 Metodologia

A metodologia utilizada neste trabalho consiste em calcular a qualidade das soluções geradas ao final de cada evento (ponto de reescalonamento). Para cada instância do problema são realizadas 10 execuções de cada algoritmo que se deseja avaliar. Para garantir uma comparação justa entre os algoritmos, as execuções de cada um deles efetuam a mesma quantidade de avaliações das funções objetivo. Neste experimento, fixou-se o valor de 12.000 avaliações. Então, calcula-se a média das métricas de qualidade obtidas para cada evento e, em seguida, essas médias são comparadas de forma pareada.

Para cada algoritmo, convencionou-se calcular as médias para os primeiros 80 eventos de cada amostra. Essa decisão deve-se à grande variação na quantidade de eventos que podem ocorrer até o final da simulação do projeto de cada instância, bem como por conveniência prática de limitações de tempo, uma vez que as instâncias maiores requerem elevado tempo de processamento. Como os eventos de cada instância ocorrem sempre na mesma ordem, entende-se que não há prejuízo em termos de comparação de algoritmos dentro da mesma instância.

A comparação dos resultados leva em consideração a significância estatística apurada pelos testes de hipótese não-paramétricos de Wilcoxon, para comparação entre duas amostras, e de Friedman, para comparação simultânea entre três ou mais amostras (DERRAC et al., 2011).

Em todos os casos foi considerado um nível de significância de 0,05.

O programa desenvolvido para o experimento recebe um arquivo de configuração no qual podem ser parametrizados: a quantidade de execuções de cada simulação, os arquivos com as configurações das instâncias a serem executadas, os algoritmos a serem executados no experimento e os parâmetros de algoritmo descritos na Seção 3.3.3.1. A análise dos resultados foi feita em Python, com o uso das bibliotecas: `SciPy`[1] para computação numérica e funções estatísticas, `pandas` para análise de dados [2], `Matplotlib`[3] para construção de gráficos e `scikit-posthocs`[4] para os testes estatísticos *post-hoc*.

4.1.1 QP1: Influência da quantidade de enxames

Antes de avaliar o algoritmo proposto, foi necessário estipular previamente alguns dos seus parâmetros, em especial o número de enxames de partículas usado na busca. Para isso, foram definidas duas configurações para o MS2MO: uma contendo 5 enxames de 200 partículas cada (`MS2MO-5s200p`) e outra com 30 enxames de 60 partículas cada (`MS2MO-30s60p`). A variação no número de partículas de cada enxame é necessária para garantir a mesma quantidade de avaliação das funções objetivo. Para cada configuração foram realizadas 10 execuções do algoritmo aplicadas à instância `sT10_dT10_E5_SK4-5`.

4.1.2 QP2: Comparação do algoritmo proposto com os algoritmos da literatura

Com o intuito de validar a hipótese de que uma abordagem de otimização baseada em múltiplos enxames de partículas pode ser efetiva quando aplicada ao DSPSP, foi realizado um comparativo entre o MS2MO e mais dois algoritmos de uso difundido da literatura sobre meta-heurísticas multiobjetivo: NSGA-II (DEB et al., 2002) e SMPSO (NEBRO et al., 2009). A ideia é comparar um algoritmo genético — mesma meta-heurística da abordagem de Shen et al. (2016) — com abordagens de PSO aplicadas ao DSPSP.

4.1.3 QP3: Influência das estratégias heurísticas dinâmicas

A inicialização das populações iniciais dos enxames que compõem o MS2MO é feita por meio da composição entre as estratégias heurísticas dinâmicas descritas na seção 3.3.3.3. Para compreender o quanto cada uma delas contribui para o desempenho do algoritmo proposto, realizou-se um comparativo entre quatro variantes do MS2MO descritas na Tabela 5. As variantes diferenciam-se pela proporção da população inicial que é obtida por meio de cada estratégia. Neste experimento, foram utilizadas as mesmas proporções usadas em (SHEN et al., 2016).

[1] https://www.scipy.org/scipylib/
[2] https://pandas.pydata.org/
[3] https://matplotlib.org/
[4] https://github.com/maximtrp/scikit-posthocs

Tabela 5 – Variantes do MS2MO

Variante	**Estratégia (%)**		
	Aleatórias	**Histórico**	**Ajuste Proativo**
MS2MO	100	0	0
MS2MO-C	50	20	30
MS2MO-H	70	20	0
MS2MO-P	70	0	30

Fonte: Próprio autor

4.1.4 Instâncias

Os experimentos consideraram dados de algumas das 18 instâncias artificiais de simulação de projetos criadas e utilizadas por Shen et al. (2016). As instâncias diferenciam-se pelas quantidades de tarefas iniciais, de novas tarefas que surgem ao longo da execução do projeto, de empregados e de habilidades por empregado. Em cada instância, há um total de 10 habilidades requeridas para o projeto e cada tarefa requer 5 diferentes habilidades escolhidas entre elas. A quantidade de empregados em um projeto pode variar entre 5, 10 e 15, sendo que o número de habilidades que cada um deles possui pode variar entre 4 e 5 em algumas instâncias ou entre 6 e 7 em outras. Entre os empregados, uma proporção de 20% trabalha em tempo parcial, com uma dedicação que varia aleatoriamente no intervalo $[0.5, 1)$. Outros 20% podem trabalhar em regime de horas extras, de modo que suas dedicações variam aleatoriamente no intervalo de $(1, 1.5]$.

Cada instância é representada por um identificador que descreve suas características, seguindo o formato `sT#1_dT#2_E#3_SK#4-#5`. Nesse formato, #1 é o número inicial de tarefas do projeto, #2 é o número de novas tarefas que surgem durante o projeto, #3 é o número de empregados disponíveis no início do projeto e #4 e #5 são o número mínimo e máximo de habilidades requeridas para uma tarefa, respectivamente. Por exemplo, o identificador $s30\_d10\_e15\_sk6-7$ denota que a instância possui 30 tarefas iniciais, então 10 novas tarefas chegam no decorrer do projeto e há um total de 15 empregados disponíveis, cada um possuindo entre 6 e 7 habilidades específicas.

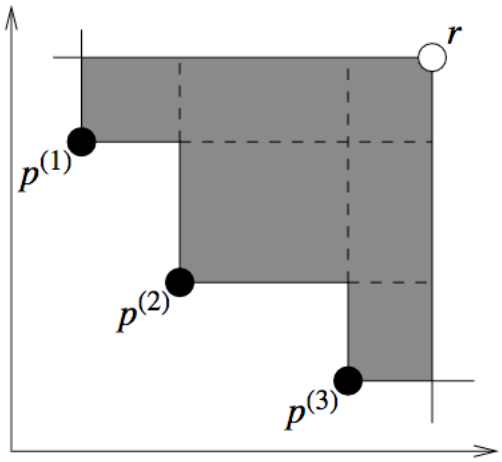
4.1.5 Métricas de Desempenho

Os resultados do experimento são avaliados pela métrica Hipervolume (FONSECA; PAQUETE; LÓPEZ-IBÁNEZ, 2006), comumente usada em trabalhos de otimização multiobjetivo. O hipervolume de uma fronteira é calculado em relação a um determinado ponto de referência no hiperplano, que deve representar uma solução ruim o suficiente para não dominar nenhuma outra que faça parte da fronteira avaliada. Sendo assim, um maior valor de hipervolume indica que o algoritmo obteve um melhor conjunto de soluções. No contexto deste trabalho, portanto, hipervolumes altos indicam um melhor conjunto de escalonamentos, considerando todos os objetivos calculados.

A Figura 6 ilustra um exemplo de como uma fronteira bidimensional $(p^{(1)}, p^{(2)}, p^{(3)})$

tem seu hipervolume obtido em relação a um ponto de referência r.

Figura 6 – Hipervolume de uma fronteira de Pareto para dois objetivos



Fonte: Adaptado de Fonseca, Paquete e López-Ibánez (2006)

Nos experimentos realizados neste trabalho, o programa gera um arquivo com os valores avaliações das funções-objetivo para cada solução da nova fronteira Pareto obtida após um reescalonamento. Esses valores são então normalizados, para que possam servir de entrada para o cálculo do hipervolume. Com a normalização, o intervalo possível de valores de cada objetivo passa a ser $[0.0, 1.0]$. Portanto, os pontos de referência escolhidos para cálculo do hipervolume ficaram sendo $(1.1, 1.1, 1.1)$ para o escalonamento inicial (não leva em conta a avaliação da estabilidade) e $(1.1, 1.1, 1.1, 1.1)$ para os reescalonamentos seguintes.

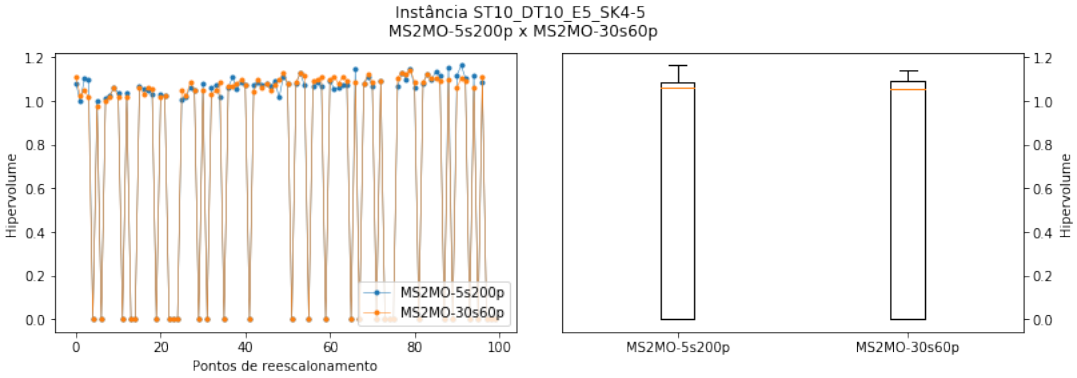
4.2 Resultados

4.2.1 Influência da quantidade de enxames

Os resultados do experimento relativo à QP1 são ilustrados pela Figura 7. Nos gráficos utilizados nas análises deste trabalho, o eixo x indica a sequência de pontos de reescalonamento do projeto, motivados pela ocorrência de eventos, e o eixo y refere-se ao valor dos hipervolumes. Cada ponto indica o valor de hipervolume obtido pelo conjunto de soluções não-dominadas geradas pela execução de um determinado algoritmo em um determinado ponto de reescalonamento.

Analisando o gráfico, podemos visualizar que os valores de hipervolume são muito parecidos em cada ponto de reescalonamento, tanto para o MS2MO com poucos enxames (`MS2MO-5s200p`), quanto para com muitos enxames (`MS2MO-30s60p`). As duas séries de valores de hipervolumes foram submetidas ao Teste de Wilcoxon, que efetuou a comparação pareada, ou seja, confrontando entre si os resultados de cada evento das duas amostras. O teste estatístico resultou em um *p-value* igual a 0,79. Considerando um nível de significância de 0,05, pode-se afirmar que não houve diferença estatística significativa entre os resultados obtidos com 5 e

Figura 7 – Hipervolumes em cada ponto de reescalonamento para o MS2MO quanto ao número de enxames



Fonte: Próprio autor

com 30 enxames de partículas, indicando que a quantidade de enxames por si só não influi no desempenho. Então, para definir qual a quantidade de enxames a ser adotada, optou-se por calcular as medianas dos hipervolumes ao longo das duas séries de eventos e então selecionar aquela que obtivesse o maior valor, como pode ser visto na Tabela 6. Segundo esse critério, foi adotada a configuração com 30 enxames, de 60 partículas cada, para os demais experimentos deste trabalho.

Tabela 6 – Medianas dos hipervolumes em cada ponto de reescalonamento para o MS2MO quanto ao número de enxames

Enxames	**Partículas**	**Mediana**
5	200	1,0571
30	60	1,0597

Fonte: Próprio autor

Queda abrupta no valor dos hipervolumes

Ao conduzir este experimento, notou-se que em alguns reescalonamentos o valor do hipervolume obtido foi igual a zero. Esse fato deve-se a características de cada instância, que pode incluir uma sequência de eventos que implique na impossibilidade de se obter uma solução que não viole nenhuma restrição. Um caso comum em instâncias de projeto com poucos empregados é a ocorrência do evento de saída de um empregado que é o único que possui determinada habilidade para execução das tarefas ativas, violando a restrição R2 (ver 3.2.7). Nesses casos, todas as soluções geradas são penalizadas, de modo a apresentar valores iguais para a função objetivo. Em casos assim, não é possível calcular o hipervolume. Portanto, convencionou-se representá-lo como sendo 0,0.

4.2.2 Comparação com outros algoritmos

O experimento relativo à QP2 tem como objetivo comparar o desempenho do MS2MO com outros algoritmos da literatura, tanto em sua formulação original, quanto com a aplicação das estratégias heurísticas dinâmicas para reinicialização das populações. Com o intuito de melhor investigar a influência das características das instâncias nos resultados, os experimentos relativos às questões de pesquisa QP2 e QP3 foram realizados com três instâncias: ST10_DT10_E5_SK4-5, para observar o ocorre quando há poucas tarefas e poucos empregados; ST10_DT10_E15_SK4-5, para avaliar como se dá o andamento do projeto com poucas tarefas e muitos empregados; e ST20_DT10_E5_SK4-5, para verificar o desempenho com muitas tarefas e poucos empregados.

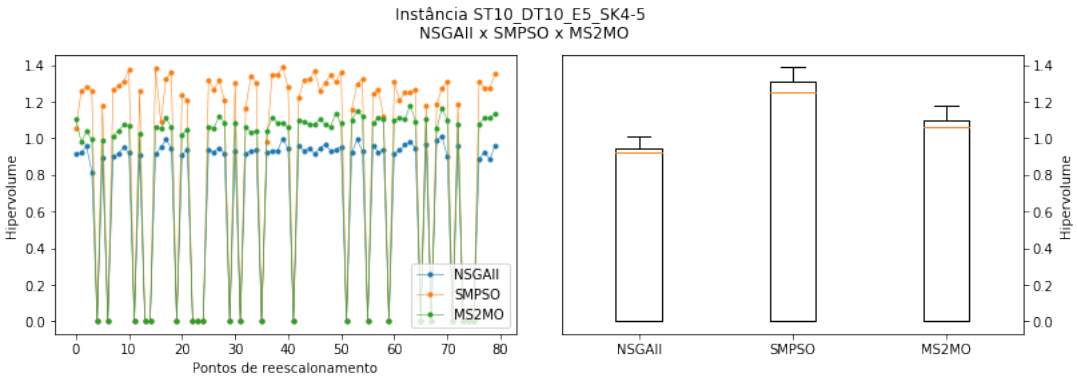
Comparação *sem* o uso de estratégias dinâmicas

Num primeiro momento, comparamos a aplicação do NSGA-II e do SMSPO em suas formulações originais com o algoritmo proposto MS2MO sem o uso de estratégias heurísticas dinâmicas. Essa comparação tem como objetivo investigar se a estratégia de múltiplos enxames é competitiva em relação aos algoritmos tradicionais.

A Figura 8 apresenta os valores dos hipervolumes a cada ponto de reescalonamento para a instância sT10_dT10_E5_SK4-5. Nela, fica nítido que o SMPSO atinge valores mais altos que os demais em praticamente todos os eventos. O MS2MO, por sua vez, somente se sobrepõe ao SMPSO em alguns poucos pontos, mas apresenta melhores resultados que o NSGA-II em todos os eventos.

Podemos notar que a queda dos valores de hipervolumes a zero ocorre sempre para os mesmos eventos e em todos os algoritmos, sugerindo que esses eventos obrigatoriamente levam à violação de alguma restrição e consequente penalização de todas as soluções geradas.

Figura 8 – Comparação entre os hipervolumes em cada ponto de reescalonamento para o NSGA-II, SMPSO e MS2MO (instância sT10_dT10_E5_SK4-5)



Fonte: Próprio autor

A comparação pareada dos *p-values* segundo o teste de Friedman (Tabela 7) confirma

as impressões da visualização dos dados, indicando que há diferença estatística significativa no desempenho dos algoritmos para a instância sT10_dT10_E5_SK4-5.

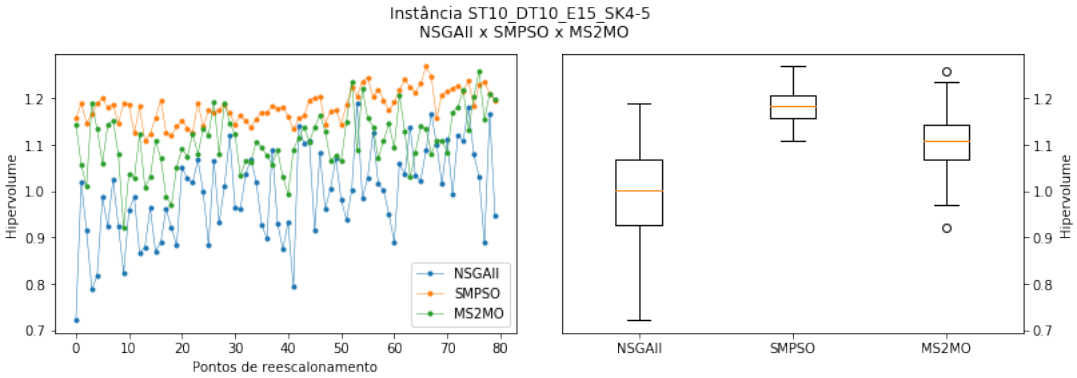
Tabela 7 – Comparação pareada dos *p-values* obtidos na execução dos algoritmos sem o uso de estratégias dinâmicas para a instância ST10_DT10_E5_SK4-5

	MS2MO	NSGAII	SMPSO
MS2MO	-1,00E+00	8,34E-06	6,37E-05
NSGAII	8,34E-06	-1,00E+00	9,19E-16
SMPSO	6,37E-05	9,19E-16	-1,00E+00

Fonte: Próprio autor

A Figura 9 apresenta os valores dos hipervolumes a cada ponto de reescalonamento para a instância sT10_dT10_E15_SK4-5. Neste caso, percebe-se que o desempenho dos algoritmos (especialmente o NSGA-II) fica mais instável entre um evento e outro, mais ainda com vantagem para o SMPSO. O MS2MO continua apresentando melhores resultados que o NSGA-II no geral, somente ficando abaixo em alguns poucos eventos.

Figura 9 – Comparação entre os hipervolumes em cada ponto de reescalonamento para o NSGA-II, SMPSO e MS2MO (instância sT10_dT10_E15_SK4-5)



Fonte: Próprio autor

Para esta instância, percebe-se que não ocorre queda dos valores de hipervolumes a zero. Isso se deve ao fato de que, como esta instância possui um grande número de empregados, torna-se muito baixa a probabilidade de que algum deles não tenha pelo menos uma habilidade requerida para as tarefas ativas, evitando a penalização das soluções.

A comparação pareada dos *p-values* segundo o teste de Friedman para a instância sT10_dT10_E15_SK4-5 (Tabela 8) confirma que há diferença estatística significativa no desempenho dos algoritmos.

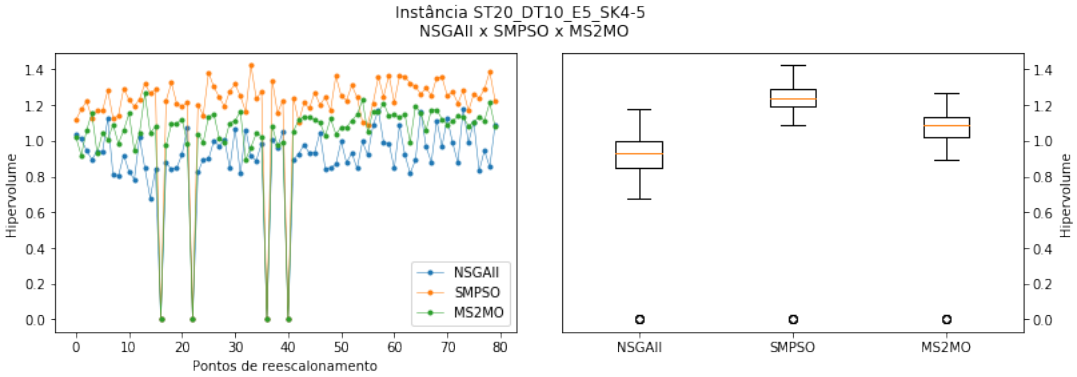
A Figura 10 apresenta os valores dos hipervolumes a cada ponto de reescalonamento para a instância sT20_dT10_E5_SK4-5. Nessa configuração de projeto com muitas tarefas e poucos empregados, repete-se o padrão de desempenho dos algoritmos, com vantagem para o SMPSO, seguido pelo MS2MO e NSGA-II.

Tabela 8 – Comparação pareada dos *p-values* obtidos na execução dos algoritmos sem o uso de estratégias dinâmicas para a instância ST10_DT10_E15_SK4-5

	MS2MO	NSGAII	SMPSO
MS2MO	-1,00E+00	1,70E-16	2,94E-18
NSGAII	1,70E-16	-1,00E+00	2,12E-47
SMPSO	2,94E-18	2,12E-47	-1,00E+00

Fonte: Próprio autor

Figura 10 – Comparação entre os hipervolumes em cada ponto de reescalonamento para o NSGA-II, SMPSO e MS2MO (instância sT20_dT10_E5_SK4-5)



Fonte: Próprio autor

Nota-se que nesta instância volta a ocorrer queda dos valores de hipervolumes a zero em alguns eventos. Como esta instância possui um pequeno número de empregados, aumenta a probabilidade de que algum deles não tenha pelo menos uma habilidade requerida para as tarefas ativas, penalizando as soluções.

A comparação pareada dos *p-values* segundo o teste de Friedman para a instância sT20_dT10_E5_SK4-5 (Tabela 9) confirma que há diferença estatística significativa no desempenho dos algoritmos.

Tabela 9 – Comparação pareada dos *p-values* obtidos na execução dos algoritmos sem o uso de estratégias dinâmicas para a instância ST20_DT10_E5_SK4-5

	MS2MO	NSGAII	SMPSO
MS2MO	-1,00E+00	7,92E-13	1,23E-20
NSGAII	7,92E-13	-1,00E+00	1,17E-45
SMPSO	1,23E-20	1,17E-45	-1,00E+00

Fonte: Próprio autor

Os resultados obtidos por esses comparativos sugerem que, mesmo sem a aplicação de estratégias dinâmicas, a abordagem PSO pode ser uma alternativa viável em relação ao NSGA-II quando aplicada ao DSPSP, ainda que o algoritmo de múltiplos enxames aqui proposto tenha

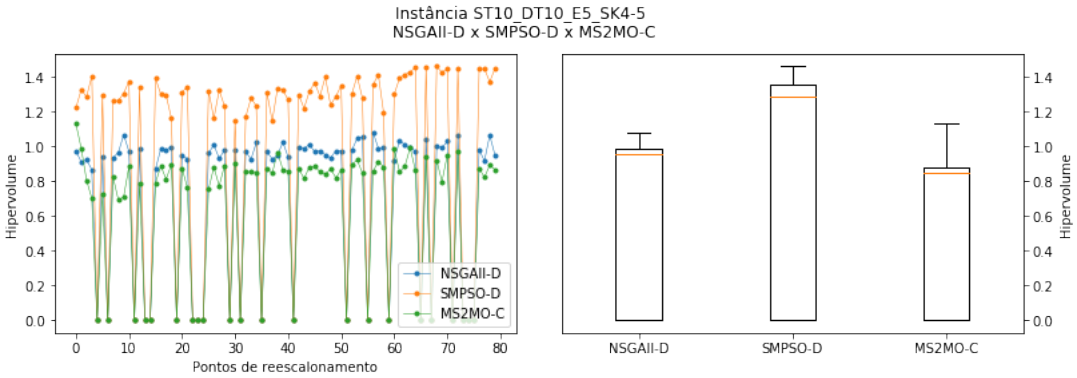
obtido desempenho inferior ao SMPSO com enxame único.

Comparação *com* o uso de estratégias dinâmicas

Em seguida é feita a comparação entre os algoritmos incorporando as estratégias dinâmicas. Aqui, chamamos as variantes dinâmicas do NSGA-II e SMPSO de NSGA-II-D e SMPSO-D, respectivamente. Estas variantes são comparadas com o MS2MO-C descrito anteriormente.

A Figura 11 apresenta os valores dos hipervolumes a cada ponto de reescalonamento para a instância sT10_dT10_E5_SK4-5. Nela, fica nítido que o SMPSO-D atinge valores bem mais altos que os demais em todos os eventos. No entanto, ao contrário de suas versões sem a aplicação de estratégicas dinâmicas, percebe-se um melhor desempenho do NSGA-II-D em relação ao MS2MO-C.

Figura 11 – Comparação entre os hipervolumes em cada ponto de reescalonamento para o NSGA-II-D, SMPSO-D e MS2MO-C (instância sT10_dT10_E5_SK4-5)



Fonte: Próprio autor

A comparação pareada dos *p-values* segundo o teste de Friedman para a instância sT10_dT10_E5_SK4-5 (Tabela 10) confirma que há diferença estatística significativa no desempenho dos algoritmos.

Tabela 10 – Comparação pareada dos *p-values* obtidos na execução dos algoritmos com o uso de estratégias dinâmicas para a instância ST10_DT10_E5_SK4-5

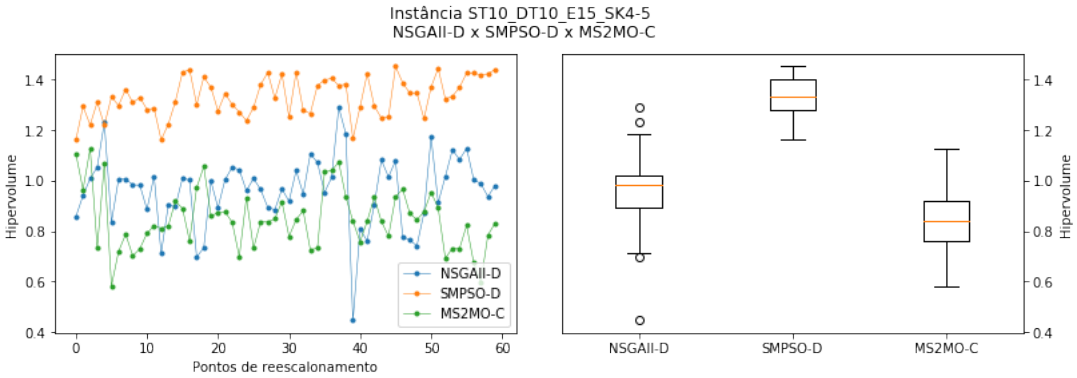
	MS2MO-C	NSGAII-D	SMPSO-D
MS2MO-C	-1,00E+00	3,34E-04	3,75E-15
NSGAII-D	3,34E-04	-1,00E+00	3,16E-06
SMPSO-D	3,75E-15	3,16E-06	-1,00E+00

Fonte: Próprio autor

A Figura 12 mostra os valores dos hipervolumes a cada ponto de reescalonamento para a instância sT10_dT10_E15_SK4-5. Mais uma vez, percebe-se a nítida vantagem do SMPSO-D

sobre os demais algoritmos, além de obter valores mais estáveis entre os eventos. Assim como na instância anterior, percebe-se um melhor desempenho do NSGA-II-D em relação ao MS2MO-C.

Figura 12 – Comparação entre os hipervolumes em cada ponto de reescalonamento para o NSGA-II-D, SMPSO-D e MS2MO-C (instância sT10_dT10_E15_SK4-5)



Fonte: Próprio autor

A comparação pareada dos *p-values* segundo o teste de Friedman para a instância sT10_dT10_E15_SK4-5 (Tabela 11) confirma a diferença estatística significativa no desempenho dos algoritmos.

Tabela 11 – Comparação pareada dos *p-values* obtidos na execução dos algoritmos com o uso de estratégias dinâmicas para a instância ST10_DT10_E15_SK4-5

	MS2MO-C	NSGAII-D	SMPSO-D
MS2MO-C	-1,00E+00	1,05E-08	1,79E-48
NSGAII-D	1,05E-08	-1,00E+00	9,82E-32
SMPSO-D	1,79E-48	9,82E-32	-1,00E+00

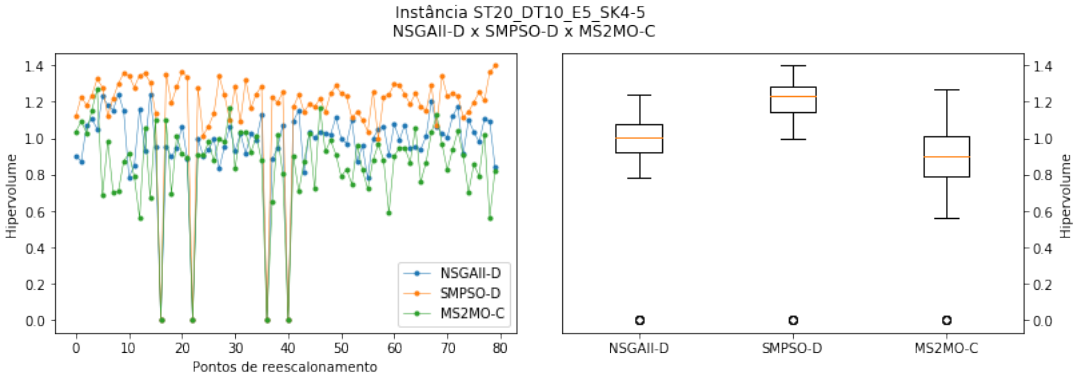
Fonte: Próprio autor

A Figura 13 apresenta os valores dos hipervolumes a cada ponto de reescalonamento para a instância sT20_dT10_E5_SK4-5. Percebe-se por ela que os valores ficam mais próximos entre si, mas ainda com nítida vantagem para o SMPSO-D. O desempenho do NSGA-II-D mais uma vez parece superar o do MS2MO-C.

A comparação pareada dos *p-values* segundo o teste de Friedman para a instância sT20_dT10_E5_SK4-5 (Tabela 12) indica que há diferença estatisticamente significativa, confirmando o melhor desempenho do NSGA-II-D sobre o MS2MO-D e do SMPSO-D sobre todos os demais.

Após os comparativos, constatou-se que o SMPSO-D apresentou melhor resultado de hipervolumes em todas as instâncias. No entanto, desta vez foi seguido pelo NSGA-II-D e depois pelo MS2MO-C. Esse resultado pode sugerir que a aplicação de estratégicas dinâmicas seja efetiva no desempenho do NSGA-II. Para investigar essa suspeita, procedemos a comparação de cada algoritmo com sua respectiva variante dinâmica separadamente.

Figura 13 – Comparação entre os hipervolumes em cada ponto de reescalonamento para o NSGA-II-D, SMPSO-D e MS2MO-C (instância sT20_dT10_E5_SK4-5)



Fonte: Próprio autor

Tabela 12 – Comparação pareada dos *p-values* obtidos na execução dos algoritmos com o uso de estratégias dinâmicas para a instância ST20_DT10_E5_SK4-5

	MS2MO-C	NSGAII-D	SMPSO-D
MS2MO-C	-1,00E+00	2,00E-05	2,04E-34
NSGAII-D	2,00E-05	-1,00E+00	3,22E-20
SMPSO-D	2,04E-34	3,22E-20	-1,00E+00

Fonte: Próprio autor

Comparação dos algoritmos com suas variantes dinâmicas

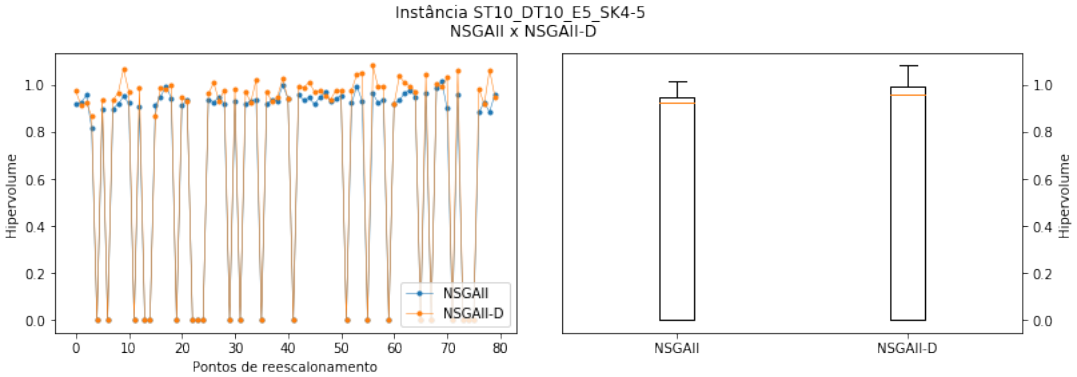
Os resultados da aplicação de estratégias dinâmicas indicam que elas podem ter melhorado o desempenho do NSGA-II para este problema. Portanto, o passo seguinte do experimento foi comparar cada algoritmo com sua variante dinâmica.

A Figura 14 sugere desempenhos semelhantes na comparação do NSGA-II com o NSGA-II-D para a instância sT10_dT10_E5_SK4-5. Verificando a média dos hipervolumes na Tabela 13, verificamos que NSGA-II-D se saiu melhor, com média de 0,696, contra 0,684 do NSGA-II. A comparação pareada usando o teste de Wilcoxon resultou em *p-value* de 4,3086E-08 confirmando que há diferença estatística significativa entre as amostras.

A Figura 15 também sugere desempenhos semelhantes na comparação do SMPSO com o SMPSO-D para a instância sT10_dT10_E5_SK4-5, com ligeira vantagem para a variante dinâmica. De fato, a média dos hipervolumes do SMPSO-D foi igual a 1,060, contra 0,975 do SMPSO. A comparação pareada usando o teste de Wilcoxon resultou em *p-value* de 1,8801E-03, confirmando a significância estatística.

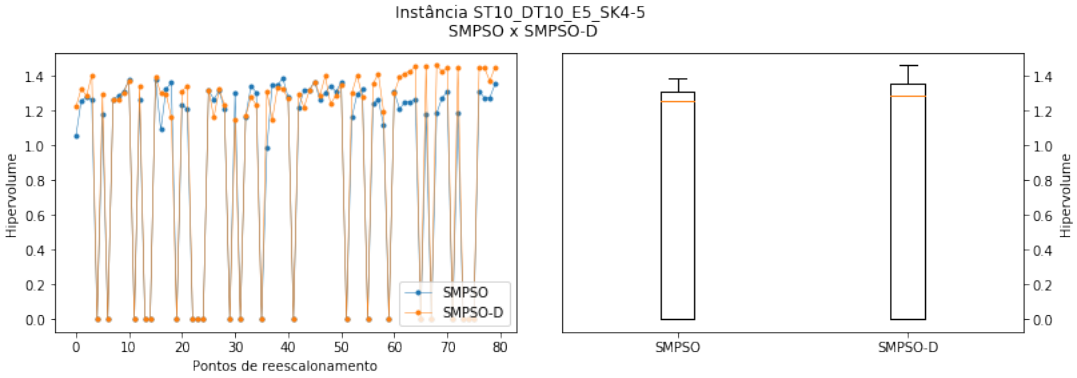
Na Figura 16 pode-se visualizar nitidamente que, ao contrário do que se supunha, o desempenho da variante dinâmica MS2MO-C foi pior que o do MS2MO sem estratégias dinâmicas, no contexto da instância sT10_dT10_E5_SK4-5. A média dos hipervolumes para o

Figura 14 – Hipervolumes em cada ponto de reescalonamento para a comparação entre NSGA-II e NSGA-II Dinâmico (instância sT10_dT10_E5_SK4-5)



Fonte: Próprio autor

Figura 15 – Hipervolumes em cada ponto de reescalonamento para a comparação entre SMPSO e SMPSO Dinâmico (instância sT10_dT10_E5_SK4-5)



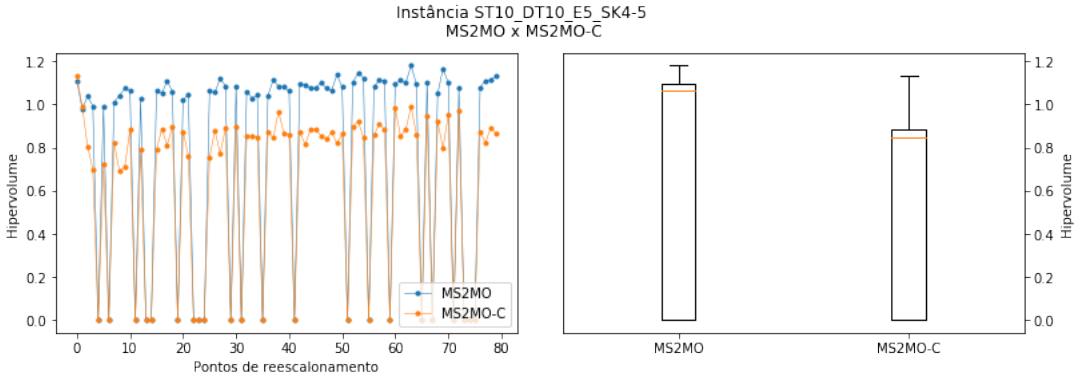
Fonte: Próprio autor

MS2MO foi de 0,790 e para o MS2MO-C foi de 0,613. A comparação pareada usando o teste de Wilcoxon resultou em *p-value* de 4,1018E-11, garantindo a significância.

A Figura 17 sugere desempenhos semelhantes na comparação do NSGA-II com o NSGA-II-D para a instância sT10_dT10_E15_SK4-5. Consultando a média dos hipervolumes na Tabela 13, vemos que para essa instância com mais empregados foi o NSGA-II se saiu melhor, com média de 0,963, contra 0,871 do NSGA-II-D. A comparação pareada usando o teste de Wilcoxon resultou em *p-value* de 4,9243E-02 confirmando que há diferença estatística significativa entre as amostras.

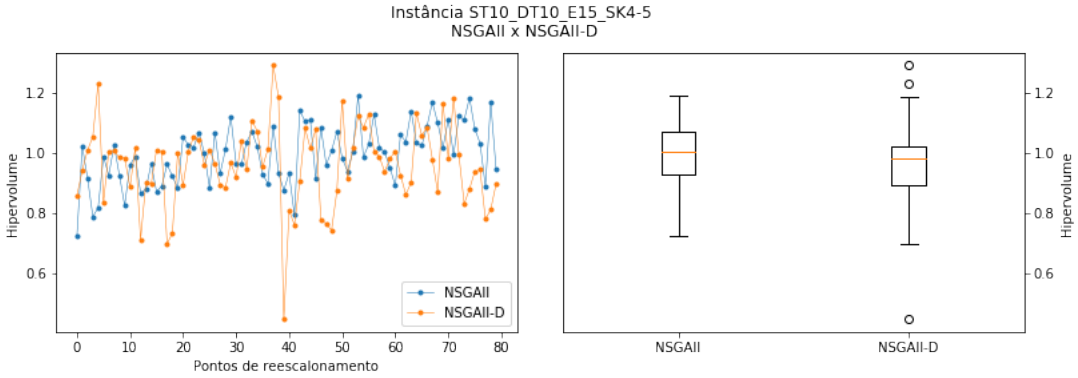
A Figura 18 indica nitidamente desempenho superior do SMPSO-D em comparação com o SMPSO para a instância sT10_dT10_E15_SK4-5. A média dos hipervolumes do SMPSO-D foi igual a 1,34 — melhor resultado entre todos os algoritmos em todas as instâncias —, contra 1,18 do SMPSO. A comparação pareada usando o teste de Wilcoxon resultou em *p-value* de 2,0977E-11, confirmando a significância estatística.

Figura 16 – Hipervolumes em cada ponto de reescalonamento para a comparação entre MS2MO e MS2MO-C (instância sT10_dT10_E5_SK4-5)



Fonte: Próprio autor

Figura 17 – Hipervolumes em cada ponto de reescalonamento para a comparação entre NSGA-II e NSGA-II Dinâmico (instância sT10_dT10_E15_SK4-5)



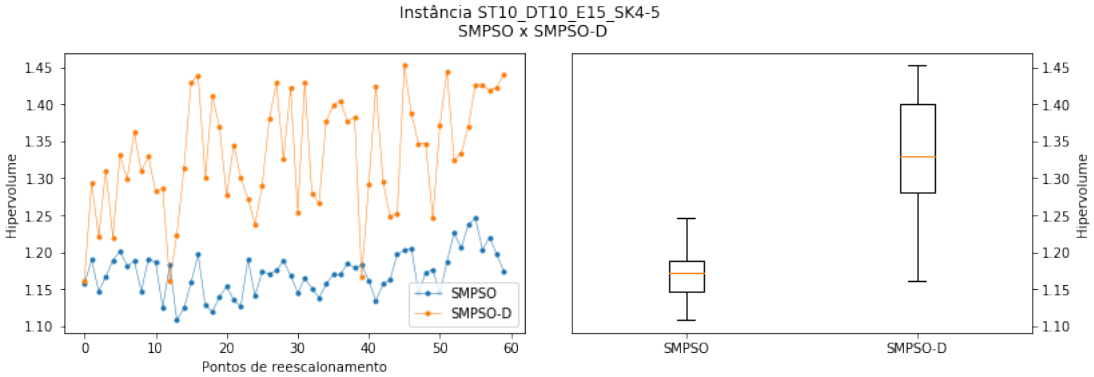
Fonte: Próprio autor

Na Figura 19 vemos que, mais uma vez, o desempenho da variante dinâmica MS2MO-C foi pior que o do MS2MO sem estratégias dinâmicas, no contexto da instância sT10_dT10_E15_SK4-5. A média dos hipervolumes para o MS2MO foi de 1,07 e para o MS2MO-C foi de 0,829. A comparação pareada usando o teste de Wilcoxon resultou em *p-value* de 7,0463E-14, garantindo a significância.

A Figura 20 sugere desempenhos semelhantes na comparação do NSGA-II com o NSGA-II-D para a instância sT20_dT10_E5_SK4-5. No entanto, consultando a média dos hipervolumes na Tabela 13, vemos que, quando a instância voltou a ter poucos empregados, o NSGA-II-D voltou a se sair melhor, com média de 1,000, contra 0,885 do NSGA-II. A comparação pareada usando o teste de Wilcoxon resultou em *p-value* de 1,1696E-03 confirmando que há diferença estatística significativa entre as amostras.

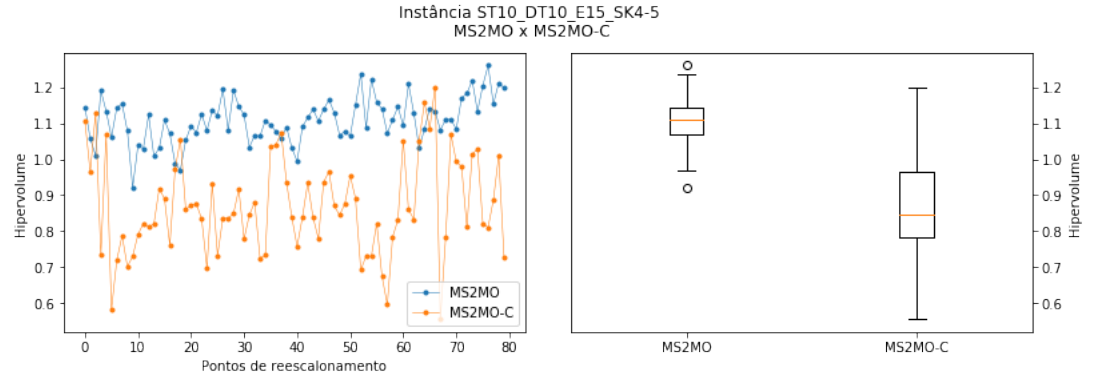
A Figura 21 indica desempenhos semelhantes do SMPSO-D em comparação com o SMPSO para a instância sT20_dT10_E5_SK4-5. Verificando a média dos hipervolumes, a do

Figura 18 – Hipervolumes em cada ponto de reescalonamento para a comparação entre SMPSO e SMPSO Dinâmico (instância sT10_dT10_E15_SK4-5)



Fonte: Próprio autor

Figura 19 – Hipervolumes em cada ponto de reescalonamento para a comparação entre MS2MO e MS2MO-C (instância sT10_dT10_E15_SK4-5)



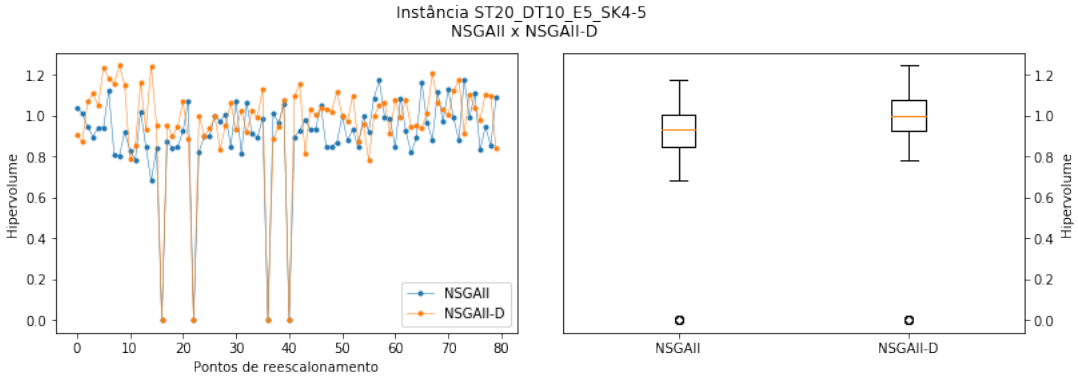
Fonte: Próprio autor

SMPSO-D foi igual a 1,16 e a do SMPSO resultou em 1,17. A comparação pareada usando o teste de Wilcoxon resultou em *p-value* de 1,8337E-01, o que significa que não houve diferença estatística entre a aplicação dos algoritmos para esta instância com muitas tarefas.

Na Figura 22 vemos que, novamente, o desempenho da variante dinâmica MS2MO-C foi pior que o do MS2MO sem estratégias dinâmicas, no contexto da instância sT20_dT10_E5_SK4-5. A média dos hipervolumes para o MS2MO foi de 1,030 e para o MS2MO-C foi de 0,865. A comparação pareada usando o teste de Wilcoxon resultou em *p-value* de 3,8050E-11, garantindo a significância.

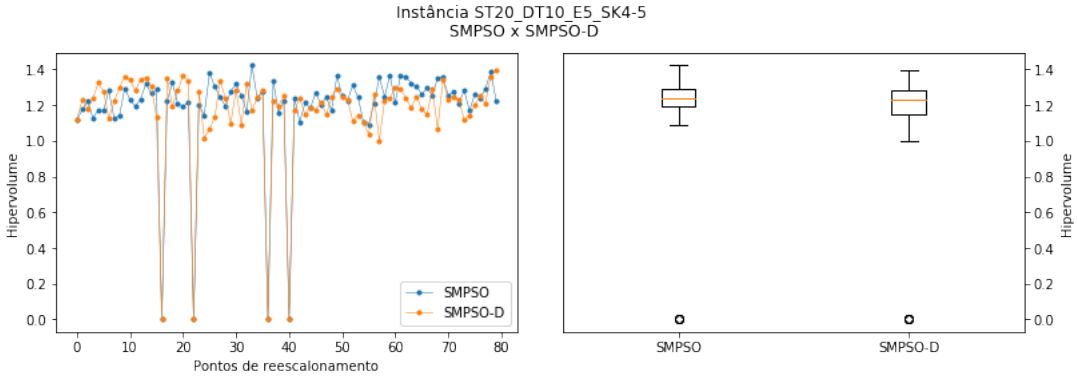
Em linhas gerais, para esta questão de pesquisa, constatou-se que quando comparamos o desempenho do NSGA-II em relação à aplicação de estratégias dinâmicas, verificamos que estas influenciam positivamente no desempenho. A influência das estratégias dinâmicas também foi significativa quando estas foram aplicadas ao SMPSO. Quanto ao algoritmo proposto MS2MO, a influência das estratégias dinâmicas também foi significativa. No entanto, ao contrário do

Figura 20 – Hipervolumes em cada ponto de reescalonamento para a comparação entre NSGA-II e NSGA-II Dinâmico (instância sT20_dT10_E5_SK4-5)



Fonte: Próprio autor

Figura 21 – Hipervolumes em cada ponto de reescalonamento para a comparação entre SMPSO e SMPSO Dinâmico (instância sT20_dT10_E5_SK4-5)

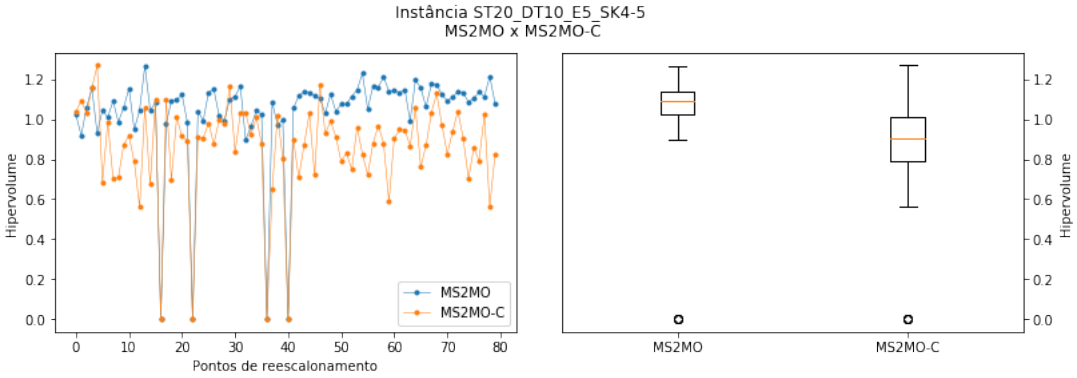


Fonte: Próprio autor

NSGA-II e do SMPSO, a variante dinâmica apresentou pior desempenho que o MS2MO sem o uso de estratégias dinâmicas.

A Tabela 13 apresenta as médias e desvios-padrão dos hipervolumes gerados por cada algoritmo em cada instância. A comparação pareada da significância estatística entre todos os algoritmos para as diferentes instâncias é ilustrada pela Figura 23 para a instância sT10_dT10_E5_SK4-5, pela Figura 24 para a instância sT10_dT10_E15_SK4-5 e pela Figura 25 para a instância sT20_dT10_E5_SK4-5.

Figura 22 – Hipervolumes em cada ponto de reescalonamento para a comparação entre MS2MO e MS2MO-C (instância sT20_dT10_E5_SK4-5)



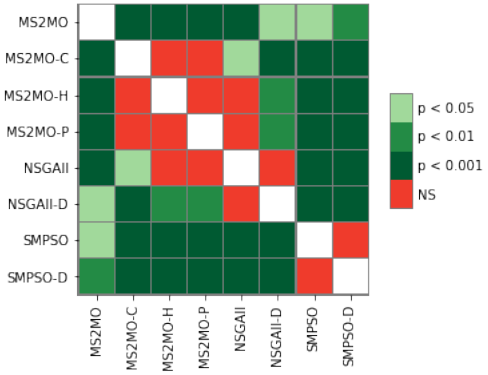
Fonte: Próprio autor

Tabela 13 – Médias e desvios-padrão dos hipervolumes gerados por cada algoritmo em cada instância

	sT10_dT10_E5_SK4-5		**sT10_dT10_E15_SK4-5**		**sT20_dT10_E5_SK4-5**	
Algoritmo	**Média**	**Desvio**	**Média**	**Desvio**	**Média**	**Desvio**
MS2MO	7,90E-01	4,86E-01	1,07E+00	1,56E-01	1,03E+00	2,63E-01
MS2MO-C	6,13E-01	4,01E-01	8,29E-01	2,03E-01	8,65E-01	2,64E-01
MS2MO-H	5,88E-01	3,98E-01	8,34E-01	1,84E-01	7,70E-01	2,42E-01
MS2MO-P	5,93E-01	3,98E-01	7,40E-01	2,20E-01	8,49E-01	2,92E-01
NSGA-II	6,84E-01	4,22E-01	9,63E-01	1,59E-01	8,85E-01	2,44E-01
NSGA-II-D	6,96E-01	4,44E-01	8,71E-01	2,42E-01	1,00E+00	2,78E-01
SMPSO	9,75E-01	5,66E-01	1,18E+00	1,10E-01	1,17E+00	3,08E-01
SMPSO-D	1,06E+00	5,60E-01	1,34E+00	7,97E-02	1,16E+00	3,05E-01

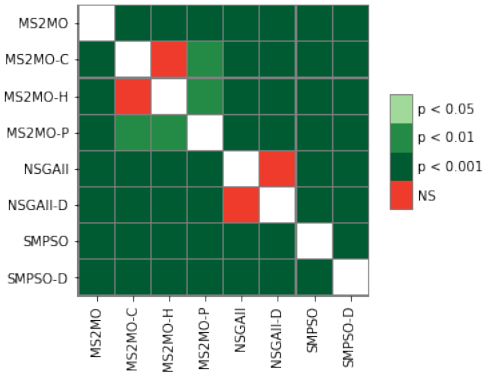
Fonte: Próprio autor

Figura 23 – Comparativo da significância indicada pelo p-value entre todos os algoritmos (instância ST10_DT10_E5_SK4-5)



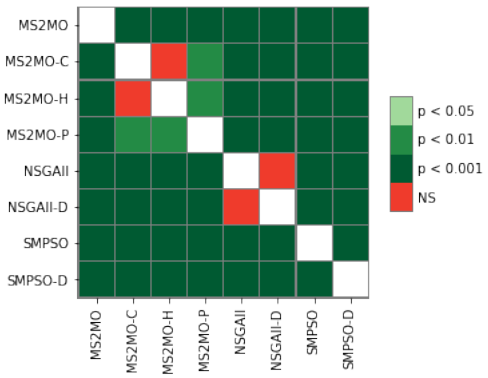
Fonte: Próprio autor

Figura 24 – Comparativo da significância indicada pelo p-value entre todos os algoritmos (instância ST10_DT10_E15_SK4-5)



Fonte: Próprio autor

Figura 25 – Comparativo da significância indicada pelo p-value entre todos os algoritmos (instância ST20_DT10_E5_SK4-5)



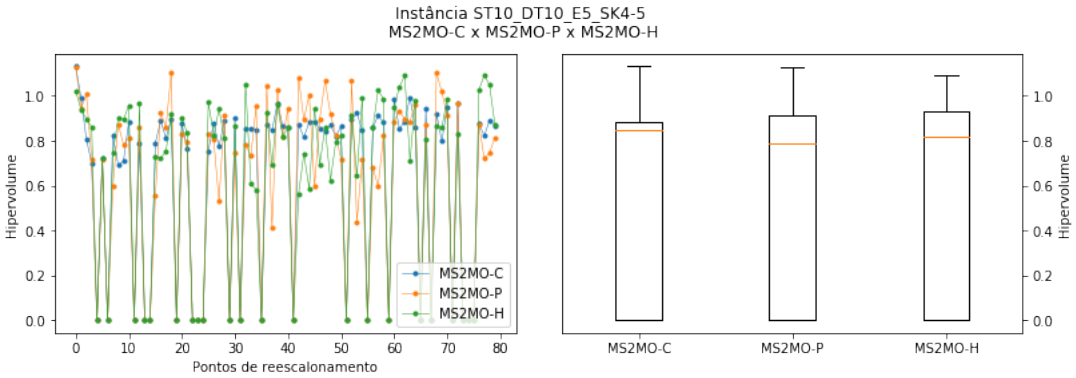
Fonte: Próprio autor

4.2.3 Influência das estratégias heurísticas dinâmicas

A finalidade da QP3 é investigar o quanto cada uma das estratégias heurísticas dinâmicas contribui para o desempenho do algoritmo proposto. Para isso foram executadas variantes do MS2MO com diferentes combinações de estratégias (ver 4.1.3).

Na Figura 26 não é possível visualizar com clareza qual algoritmo se sobressai quando aplicado à instância ST10_DT10_E5_SK4-5. Consultando as médias dos hipervolumes na Tabela 13 vemos que o MS2MO-C tem 0,613, o MS2MO-H tem 0,588 e o MS2MO-P aparece com 0,593.

Figura 26 – Comparação entre os hipervolumes em cada ponto de reescalonamento para o MS2MO-C, MS2MO-H e MS2MO-P (instância ST10_DT10_E5_SK4-5)



Fonte: Próprio autor

No entanto, a comparação pareada do teste de Friedman (Tabela 14) indica que, estatisticamente, não houve diferença entre a aplicação das três variantes do MS2MO.

Tabela 14 – Comparação pareada dos *p-values* obtidos na execução das variantes do MS2MO para a instância ST10_DT10_E5_SK4-5

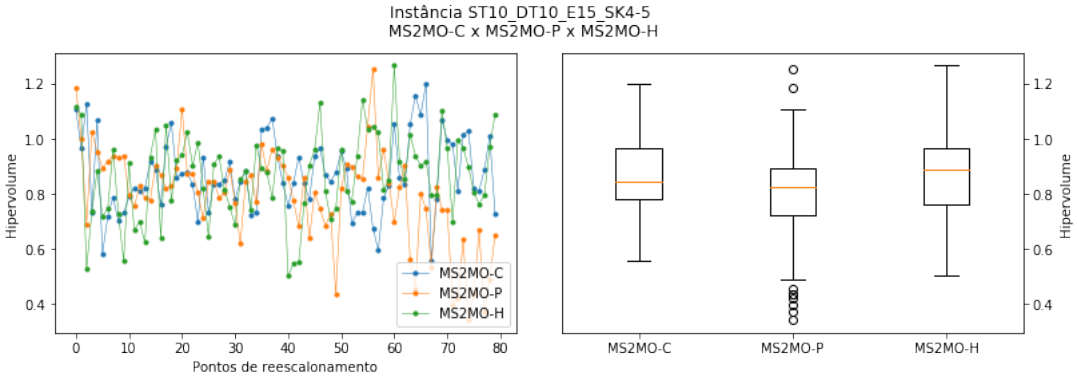
	MS2MO-C	MS2MO-H	MS2MO-P
MS2MO-C	-1,00E+00	7,90E-01	9,46E-01
MS2MO-H	7,90E-01	-1,00E+00	7,38E-01
MS2MO-P	9,46E-01	7,38E-01	-1,00E+00

Fonte: Próprio autor

Na Figura 27 também não é possível visualizar com clareza qual algoritmo se sobressai quando aplicado à instância ST10_DT10_E15_SK4-5. Verificando as médias, temos que o MS2MO-C apresentou média de 0,829, o MS2MO-H obteve 0,834 e o MS2MO-P teve 0,740.

De fato, a comparação pareada do teste de Friedman (Tabela 15) indica que o MS2MO-P, que aplica a somente a estratégia de ajuste proativo, deve desempenho levemente inferior às demais. A diferença entre MS2MO-C e MS2MO-H não foi significativa para essa instância.

Figura 27 – Comparação entre os hipervolumes em cada ponto de reescalonamento para o MS2MO-C, MS2MO-H e MS2MO-P (instância ST10_DT10_E15_SK4-5)



Fonte: Próprio autor

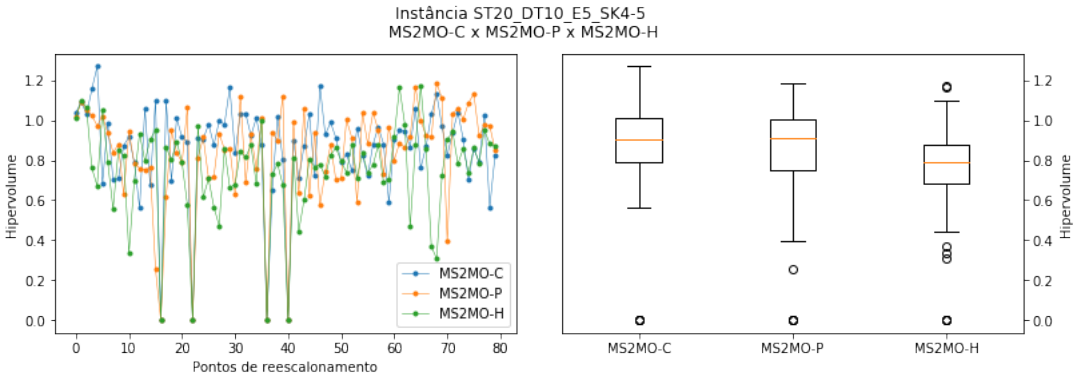
Tabela 15 – Comparação pareada dos *p-values* obtidos na execução das variantes do MS2MO para a instância ST10_DT10_E15_SK4-5

	MS2MO-C	MS2MO-H	MS2MO-P
MS2MO-C	-1,00E+00	9,19E-01	1,57E-02
MS2MO-H	9,19E-01	-1,00E+00	1,19E-02
MS2MO-P	1,57E-02	1,19E-02	-1,00E+00

Fonte: Próprio autor

Mais uma vez, na Figura 28 não se pode visualizar claramente qual algoritmo tem melhor desempenho quando aplicado à instância ST20_DT10_E5_SK4-5. Verificando as médias, temos que o MS2MO-C apresentou média de 0,865, o MS2MO-H obteve 0,770 e o MS2MO-P teve 0,849.

Figura 28 – Comparação entre os hipervolumes em cada ponto de reescalonamento para o MS2MO-C, MS2MO-H e MS2MO-P (instância ST20_DT10_E5_SK4-5)



Fonte: Próprio autor

A comparação pareada do teste de Friedman (Tabela 16) indica que, dessa vez, foi o MS2MO-H que obteve desempenho significativamente inferior aos demais, não havendo

diferença entre o MS2MO-C e o MS2MO-P.

Tabela 16 – Comparação pareada dos *p-values* obtidos na execução das variantes do MS2MO para a instância ST20_DT10_E5_SK4-5

	MS2MO-C	MS2MO-H	MS2MO-P
MS2MO-C	-1,00E+00	1,06E-04	6,97E-01
MS2MO-H	1,06E-04	-1,00E+00	4,58E-04
MS2MO-P	6,97E-01	4,58E-04	-1,00E+00

Fonte: Próprio autor

4.3 Ameaças à Validade

Durante o desenvolvimento deste trabalho, foram identificadas algumas ameaças à validade dos experimentos. Devido ao alto custo computacional para realizar as simulações, foi preciso reduzir a quantidade de a quantidade de execuções de cada algoritmo para cada instância, o que pode ter influenciado a precisão dos resultados. Pelo mesmo motivo, poderia ter sido adotada uma maior quantidade de iterações para avaliação da função objetivo, o que aumentaria a qualidade das soluções candidatas obtidas. A escolha das instâncias utilizadas no experimento é outra ameaça à validade. As instâncias escolhidas dentre as disponíveis podem ter sido pouco representativas, influenciando a validade externa do experimento, uma vez que não se sabe o quanto o desempenho dos algoritmos avaliados é suscetível a situações específicas de uma simulação de projeto.

5 Conclusão

Nesta dissertação de mestrado, foi investigada a aplicação de algoritmos de otimização multiobjetivo ao problema de escalonamento de projetos de software. Após identificadas algumas lacunas ou limitações no que se refere às pesquisas sobre SPSP, verificou-se que as modelagens existentes não levavam em conta as características dinâmicas dos projetos. Portanto, buscou-se implementar um modelo mais fiel a um ambiente real de projetos. Para isso, decidiu-se por uma abordagem baseada no trabalho de Shen et al. (2016), que considera as necessidades de reescalonamento dos cronogramas de acordo com a ocorrência de eventos no projeto, tais como mudanças na composição da equipe e a chegada de novas tarefas.

Dessa forma, foi desenvolvido o *framework* `spsp-jmetal`[1] para experimentos em SPSP baseado na biblioteca jMetal, com a finalidade de aproveitar os algoritmos de otimização multiobjetivo já implementados pela biblioteca. Adicionalmente, foi desenvolvido o MS2MO, algoritmo baseado em múltiplos enxames de partículas, como forma de investigar as possibilidades dessa meta-heurística em problemas de otimização dinâmica. O desempenho do MS2MO foi comparado aos do NSGA-II e do SMPSO, visto que estas abordagens ainda não haviam sido aplicadas ao problema proposto. Também verificou-se a influência do uso de estratégias dinâmicas em cada um dos algoritmos.

Os experimentos sugeriram que a meta-heurística de PSO pode ser promissora em relação a um algoritmo genético como o NSGA-II, quando aplicada ao problema de escalonamento dinâmico de projetos. O algoritmo SMPSO foi o que apresentou melhor desempenho em todas as instâncias avaliadas. O algoritmo de múltiplos enxames proposto obteve desempenho mediano, chegando a ser superado pelo NSGA-II em algumas ocasiões.

Os resultados deste trabalho revelaram algumas oportunidades de melhoria e sugestões de trabalhos futuros. Entre elas estão: a) avaliação dos algoritmos em mais instâncias do problema, a fim de completar a análise dos resultados em uma variedade mais ampla de cenários; b)

[1] https://github.com/rodrigoamaral/spsp-jmetal

incorporação de outros métodos de arquivamento ao MS2MO; c) uso de outras variantes do PSO para executar os enxames do MS2MO; d) investigação sobre outras características de projetos de software que possam ser agregadas ao modelo, tornando-o ainda mais fiel ao ambiente de projetos reais; e) inclusão de outras métricas de desempenho além do hipervolume, tais como *Generational Distance* (GD) e *Inverted Generational Distance* (IGD) para verificar o desempenho quanto à convergência e à diversidade, f) análise do desempenho em termos dos valores dos objetivos de duração, custo, robustez e estabilidade; e g) refatoração do código do spsp-jmetal para otimizar o desempenho da execução do modelo, reduzindo o tempo necessário para efetuar futuros experimentos.

Referências

ALBA, E.; CHICANO, F. Searching for liveness property violations in concurrent systems with aco. In: ACM. *Proceedings of the 10th annual conference on Genetic and evolutionary computation*. [S.l.], 2008. p. 1727–1734. Citado na página 21.

ALBA, E.; CHICANO, J. Software project management with GAs. *Information Sciences*, v. 177, n. 11, p. 2380–2401, 2007. ISSN 00200255. Citado 8 vezes nas páginas 16, 23, 24, 25, 26, 27, 29 e 39.

ALVAREZ-VALDES, R. et al. A scatter search algorithm for project scheduling under partially renewable resources. *Journal of Heuristics*, v. 12, n. 1-2, p. 95–113, 2006. ISSN 13811231. Citado na página 27.

ANTONIOL, G.; Di Penta, M.; HARMAN, M. A robust search-based approach to project management in the presence of abandonment, rework, error and uncertainty. *Proceedings - International Software Metrics Symposium*, p. 172–183, 2004. ISSN 15301435. Citado 2 vezes nas páginas 26 e 27.

ANTONIOL, G.; PENTA, M. D.; HARMAN, M. Search-based techniques applied to optimization of project planning for a massive maintenance project. *IEEE International Conference on Software Maintenance, ICSM*, v. 2005, p. 240–252, 2005. ISSN 1063-6773. Citado 2 vezes nas páginas 26 e 27.

BRITTO, A.; MOSTAGHIM, S.; POZO, A. Iterated multi-swarm: A multi-swarm algorithm based on archiving methods. In: *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*. New York, NY, USA: ACM, 2013. (GECCO '13), p. 583–590. ISBN 978-1-4503-1963-8. Citado na página 17.

CARVALHO, A. B. *Novas Estratégias para Otimização por Nuvem de Partículas Aplicadas a Problemas com Muitos Objetivos*. Tese (Doutorado) — Universidade Federal do Paraná, 2013. Citado 3 vezes nas páginas 9, 30 e 31.

CHANG, C. et al. Spmnet: a formal methodology for software management. In: IEEE. *Computer Software and Applications Conference, 1994. COMPSAC 94. Proceedings., Eighteenth Annual International*. [S.l.], 1994. p. 57. Citado na página 15.

CHANG, C. K.; CHRISTENSEN, M. J.; ZHANG, T. Genetic Algorithms for Project Management. *Annals of Software Engineering*, v. 11, n. 1, p. 107–139, 2001. ISSN 10227091. Citado na página 26.

CHAO, C. *Software Project Management Net: A New Methodology on Software Management*. Tese (Doutorado), Chicago, IL, USA, 1995. UMI Order No. GAX95-32380. Citado na página 26.

CHEN, W.-N.; ZHANG, J. Ant Colony Optimization for Software Project Scheduling and Staffing with an Event-Based Scheduler. *IEEE Transactions on Software Engineering*, v. 39, n. 1, p. 1–17, 2013. ISSN 0098-5589. Citado na página 28.

CHICANO, F. et al. Using Multi-objective Metaheuristics to Solve the Software Project Scheduling Problem. *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO '11)*, p. 1915–1922, 2011. Citado na página 27.

COELLO, C. A. C. et al. *Evolutionary algorithms for solving multi-objective problems*. [S.l.]: Springer, 2007. v. 5. Citado 2 vezes nas páginas 29 e 47.

Coello Coello, C. A.; REYES-SIERRA, M. Multi-Objective Particle Swarm Optimizers: A Survey of the State-of-the-Art. *International Journal of Computational Intelligence Research*, v. 2, n. 3, 2006. ISSN 09741259. Disponível em: <http://www.ijcir.com/publishedPapers.php?showDetails=Y{&}idArticle=68{&}volume=1{&}number=1{&}volume{_}i>. Citado na página 48.

COLANZI, T. E. et al. Search Based Software Engineering: Review and analysis of the field in Brazil. *Journal of Systems and Software*, v. 86, n. 4, p. 970–984, 2013. ISSN 01641212. Citado na página 21.

CRUZ, C.; GONZÁLEZ, J. R.; PELTA, D. A. Optimization in dynamic environments: A survey on problems, methods and measures. *Soft Computing*, v. 15, n. 7, p. 1427–1448, 2011. ISSN 14327643. Citado na página 32.

DEB, K. et al. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, IEEE, v. 6, n. 2, p. 182–197, 2002. Citado 2 vezes nas páginas 47 e 55.

DERRAC, J. et al. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, Elsevier, v. 1, n. 1, p. 3–18, 2011. Citado na página 54.

Di Penta, M.; HARMAN, M.; ANTONIOL, G. The use of search-based optimization techniques to schedule and staff software projects: an approach and an empirical study. *Software: Practice and Experience*, John Wiley & Sons, Ltd., v. 41, n. 5, p. 495–519, 2011. ISSN 1097-024X. Citado na página 28.

DURILLO, J. J.; NEBRO, A. J. jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, v. 42, p. 760–771, 2011. ISSN 0965-9978. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0965997811001219>. Citado na página 35.

FERRUCCI, F. et al. Using evolutionary based approaches to estimate software development effort. *Evolutionary computation and optimization algorithms in software engineering: applications and techniques. IGI Global, Hershey, PA*, p. 13–28, 2010. Citado na página 27.

FERRUCCI, F.; HARMAN, M.; SARRO, F. Search-based software project management. In: *Project Management in a Changing World*. [S.l.: s.n.], 2014. p. 373–399. ISBN 9783642550348. Citado 4 vezes nas páginas 15, 20, 23 e 28.

FONSECA, C. M.; PAQUETE, L.; LÓPEZ-IBÁNEZ, M. An improved dimension-sweep algorithm for the hypervolume indicator. In: IEEE. *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*. [S.l.], 2006. p. 1157–1163. Citado 2 vezes nas páginas 56 e 57.

FÜLÖP, J. Introduction to decision making methods. *Laboratory of operations research and decision systems, Computer and automation institute, Hungarian Academy of Sciences*, Citeseer, v. 1, 2005. Citado na página 46.

HARMAN, M.; JONES, B. F. Search-based software engineering. *Information and Software Technology*, v. 43, n. 14, p. 833–839, 2001. ISSN 09505849. Citado 2 vezes nas páginas 15 e 20.

HARMAN, M.; MANSOURI, S. A.; ZHANG, Y. Search based software engineering: A comprehensive analysis and review of trends techniques and applications. *Department of Computer Science, King's College London, Tech. Rep. TR-09-03*, 2009. Citado na página 22.

HARMAN, M.; MANSOURI, S. A.; ZHANG, Y. Search-based Software Engineering: Trends, Techniques and Applications. *ACM Comput. Surv.*, v. 45, n. 1, p. 11:1–11:61, 2012. ISSN 0360-0300. Citado 3 vezes nas páginas 21, 22 e 27.

HERIČKO, M.; ŽIVKOVIČ, A.; ROZMAN, I. An approach to optimizing software development team size. *Information Processing Letters*, v. 108, n. 3, p. 101–106, 2008. ISSN 00200190. Citado na página 27.

ISHIBUCHI, H. et al. Behavior of EMO algorithms on many-objective optimization problems with correlated objectives. In: *2011 IEEE Congress on Evolutionary Computation (CEC 2011)*. [S.l.: s.n.], 2011. p. 1465–1472. ISSN Pending. Citado na página 16.

KANG, D.; JUNG, J.; BAE, D.-H. Constraint-based human resource allocation in software projects. *Software: Practice and Experience*, John Wiley & Sons, Ltd., v. 41, n. 5, p. 551–577, 2011. ISSN 1097-024X. Citado na página 27.

KATZ, G.; PELED, D. Model checking-based genetic programming with an application to mutual exclusion. In: SPRINGER. *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. [S.l.], 2008. p. 141–156. Citado na página 21.

KENNEDY, R. J. and eberhart, particle swarm optimization. In: *Proceedings of IEEE International Conference on Neural Networks IV, pages*. [S.l.: s.n.], 1995. v. 1000. Citado na página 47.

LUNA, F. et al. The software project scheduling problem: A scalability analysis of multi-objective metaheuristics. *Applied Soft Computing Journal*, v. 15, p. 136–148, 2014. ISSN 15684946. Citado na página 27.

MAHANTI, P.; BANERJEE, S. Automated testing in software engineering: using ant colony and self-regulated swarms. In: *Proceedings of the 17th IASTED international conference on Modelling and simulation (MS'06)*. [S.l.: s.n.], 2006. p. 443–448. Citado na página 21.

MATOS, J. L.; BRITTO, A. Multi-swarm algorithm based on archiving and topologies for many-objective optimization. In: *2017 IEEE Congress on Evolutionary Computation (CEC)*. [S.l.: s.n.], 2017. p. 1877–1884. Citado 2 vezes nas páginas 18 e 48.

MERKLE, D.; MIDDENDORF, M.; SCHMECK, H. Ant colony optimization for resource-constrained project scheduling. *IEEE Transactions on Evolutionary Computation*, v. 6, n. 4, p. 333–346, 2002. ISSN 1089-778X. Citado na página 23.

NEBRO, A. et al. SMPSO: A new pso-based metaheuristic for multi-objective optimization. *2009 IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making, MCDM 2009 - Proceedings*, n. 2, 2009. Citado 2 vezes nas páginas 49 e 55.

NEBRO, A. J. et al. A study of convergence speed in multi-objective metaheuristics. In: SPRINGER. *International Conference on Parallel Problem Solving from Nature*. [S.l.], 2008. p. 763–772. Citado 2 vezes nas páginas 47 e 49.

NGUYEN, T.; YANG, S.; BRANKE, J. Evolutionary dynamic optimization: A survey of the state of the art. *Swarm and Evolutionary Computation*, v. 6, p. 1–24, 2012. ISSN 22106502. Citado 3 vezes nas páginas 16, 17 e 32.

PEIXOTO, D. C. C.; MATEUS, G. R.; RESENDE, R. F. The issues of solving staffing and scheduling problems in software development projects. In: *2014 IEEE 38th Annual Computer Software and Applications Conference*. [S.l.: s.n.], 2014. p. 1–10. Citado na página 23.

PMI. *A Guide To The Project Management Body Of Knowledge (PMBOK Guides)*. [S.l.]: Project Management Institute, 2004. ISBN 193069945X, 9781933890517. Citado na página 22.

PRESSMAN, R. S. *Software engineering: a practitioner's approach*. [S.l.]: Palgrave Macmillan, 2005. Citado na página 15.

RÄIHÄ, O. A survey on search-based software design. *Computer Science Review*, v. 4, n. 4, p. 203–249, 2010. ISSN 15740137. Citado na página 21.

SAATY, T. L.; VARGAS, L. G. Comparison of eigenvalue, logarithmic least squares and least squares methods in estimating ratios. *Mathematical modelling*, Elsevier, v. 5, n. 5, p. 309–324, 1984. Citado na página 46.

SARRO, F. Search-based approaches for software development effort estimation. *Proceedings of the 12th International Conference on Product Focused Software Development and Process Improvement - Profes '11*, p. 38–43, 2011. Citado na página 27.

SCHÜTZE, O.; LARA, A.; COELLO, C. A. C. On the influence of the number of objectives on the hardness of a multiobjective optimization problem. *IEEE Transactions Evolutionary Computation*, v. 15, n. 4, p. 444–455, 2011. Citado na página 16.

SHAW, M. Prospects for an engineering discipline of software. *IEEE Software*, v. 7, n. 6, p. 15–24, Nov 1990. ISSN 0740-7459. Citado na página 15.

SHEN, X. et al. Dynamic Software Project Scheduling through a Proactive-rescheduling Method. *IEEE Transactions on Software Engineering*, v. 42, n. 7, p. 1–1, 2016. ISSN 0098-5589. Citado 16 vezes nas páginas 16, 17, 18, 29, 35, 36, 37, 40, 45, 46, 51, 52, 53, 55, 56 e 74.

SRINIVAS, N.; DEB, K. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, MIT Press, v. 2, n. 3, p. 221–248, 1994. Citado na página 47.

WU, X. et al. An Evolutionary Hyper-heuristic for the Software Project Scheduling Problem. In: SPRINGER. *International Conference on Parallel Problem Solving from Nature*. [S.l.], 2016. p. 37–47. Citado na página 28.

XIAO, J. et al. Dynamic Resource Scheduling in Disruption-Prone Software Development Environments. In: *13th international conference on Fundamental Approaches to Software Engineering*. [S.l.: s.n.], 2010. v. 6013, p. 107–122. Citado 2 vezes nas páginas 16 e 28.